

# What Is ClickHouse?

ClickHouse® is a column-oriented database management system (DBMS) for online analytical processing of queries (OLAP).

In a “normal” row-oriented DBMS, data is stored in this order:

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

In other words, all the values related to a row are physically stored next to each other.

Examples of a row-oriented DBMS are MySQL, Postgres, and MS SQL Server.

In a column-oriented DBMS, data is stored like this:

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

These examples only show the order that data is arranged in. The values from different columns are stored separately, and data from the same column is stored together.

Examples of a column-oriented DBMS: Vertica, Paraccel (Actian Matrix and Amazon Redshift), Sybase IQ, Exasol, Infobright, InfiniDB, MonetDB (VectorWise and Actian Vector), LucidDB, SAP HANA, Google Dremel, Google PowerDrill, Druid, and kdb+.

Different orders for storing data are better suited to different scenarios. The data access scenario refers to what queries are made, how often, and in what proportion; how much data is read for each type of query – rows, columns, and bytes; the relationship between reading and updating data; the working size of the data and how locally it is used; whether transactions are used, and how isolated they are; requirements for data replication and logical integrity; requirements for latency and throughput for each type of query, and so on.

The higher the load on the system, the more important it is to customize the system set up to match the requirements of the usage scenario, and the more fine grained this customization becomes. There is no system that is equally well-suited to significantly different scenarios. If a system is adaptable to a wide set

of scenarios, under a high load, the system will handle all the scenarios equally poorly, or will work well for just one or few of possible scenarios.

## Key Properties of OLAP Scenario

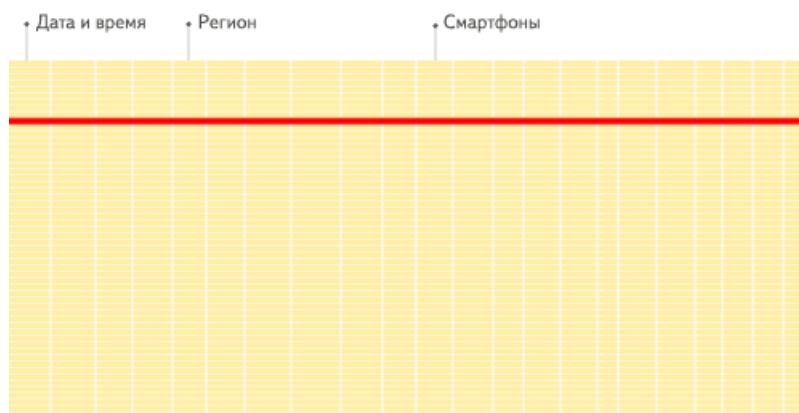
- The vast majority of requests are for read access.
- Data is updated in fairly large batches ( $> 1000$  rows), not by single rows; or it is not updated at all.
- Data is added to the DB but is not modified.
- For reads, quite a large number of rows are extracted from the DB, but only a small subset of columns.
- Tables are “wide,” meaning they contain a large number of columns.
- Queries are relatively rare (usually hundreds of queries per server or less per second).
- For simple queries, latencies around 50 ms are allowed.
- Column values are fairly small: numbers and short strings (for example, 60 bytes per URL).
- Requires high throughput when processing a single query (up to billions of rows per second per server).
- Transactions are not necessary.
- Low requirements for data consistency.
- There is one large table per query. All tables are small, except for one.
- A query result is significantly smaller than the source data. In other words, data is filtered or aggregated, so the result fits in a single server’s RAM.

It is easy to see that the OLAP scenario is very different from other popular scenarios (such as OLTP or Key-Value access). So it does not make sense to try to use OLTP or a Key-Value DB for processing analytical queries if you want to get decent performance. For example, if you try to use MongoDB or Redis for analytics, you will get very poor performance compared to OLAP databases.

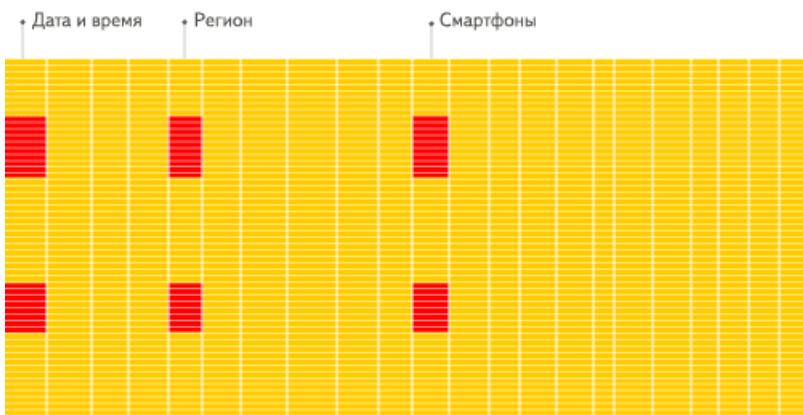
## Why Column-Oriented Databases Work Better in the OLAP Scenario

Column-oriented databases are better suited to OLAP scenarios: they are at least 100 times faster in processing most queries. The reasons are explained in detail below, but the fact is easier to demonstrate visually:

### Row-oriented DBMS



### Column-oriented DBMS



See the difference?

## Input/output

1. For an analytical query, only a small number of table columns need to be read. In a column-oriented database, you can read just the data you need. For example, if you need 5 columns out of 100, you can expect a 20-fold reduction in I/O.
2. Since data is read in packets, it is easier to compress. Data in columns is also easier to compress. This further reduces the I/O volume.
3. Due to the reduced I/O, more data fits in the system cache.

For example, the query “count the number of records for each advertising platform” requires reading one “advertising platform ID” column, which takes up 1 byte uncompressed. If most of the traffic was not from advertising platforms, you can expect at least 10-fold compression of this column. When using a quick compression algorithm, data decompression is possible at a speed of at least several gigabytes of uncompressed data per second. In other words, this query can be processed at a speed of approximately several billion rows per second on a single server. This speed is actually achieved in practice.

## CPU

Since executing a query requires processing a large number of rows, it helps to dispatch all operations for entire vectors instead of for separate rows, or to implement the query engine so that there is almost no dispatching cost. If you do not do this, with any half-decent disk subsystem, the query interpreter inevitably stalls the CPU. It makes sense to both store data in columns and process it, when possible, by columns.

There are two ways to do this:

1. A vector engine. All operations are written for vectors, instead of for separate values. This means you do not need to call operations very often, and dispatching costs are negligible. Operation code contains an optimized internal cycle.
2. Code generation. The code generated for the query has all the indirect calls in it.

This is not done in “normal” databases, because it does not make sense when running simple queries. However, there are exceptions. For example, MemSQL uses code generation to reduce latency when processing SQL queries. (For comparison, analytical DBMSs require optimization of throughput, not latency.)

Note that for CPU efficiency, the query language must be declarative (SQL or MDX), or at least a vector (J, K). The query should only contain implicit loops, allowing for optimization.

---

# Distinctive Features of ClickHouse

# True Column-Oriented Database Management System

In a real column-oriented DBMS, no extra data is stored with the values. Among other things, this means that constant-length values must be supported, to avoid storing their length “number” next to the values. For example, a billion UInt8-type values should consume around 1 GB uncompressed, or this strongly affects the CPU use. It is essential to store data compactly (without any “garbage”) even when uncompressed since the speed of decompression (CPU usage) depends mainly on the volume of uncompressed data.

It is worth noting because there are systems that can store values of different columns separately, but that can't effectively process analytical queries due to their optimization for other scenarios. Examples are HBase, BigTable, Cassandra, and HyperTable. You would get throughput around a hundred thousand rows per second in these systems, but not hundreds of millions of rows per second.

It's also worth noting that ClickHouse is a database management system, not a single database. ClickHouse allows creating tables and databases in runtime, loading data, and running queries without reconfiguring and restarting the server.

## Data Compression

Some column-oriented DBMSs do not use data compression. However, data compression does play a key role in achieving excellent performance.

In addition to efficient general-purpose compression codecs with different trade-offs between disk space and CPU consumption, ClickHouse provides [specialized codecs](#) for specific kinds of data, which allow ClickHouse to compete with and outperform more niche databases, like time-series ones.

## Disk Storage of Data

Keeping data physically sorted by primary key makes it possible to extract data for its specific values or value ranges with low latency, less than a few dozen milliseconds. Some column-oriented DBMSs (such as SAP HANA and Google PowerDrill) can only work in RAM. This approach encourages the allocation of a larger hardware budget than is necessary for real-time analysis.

ClickHouse is designed to work on regular hard drives, which means the cost per GB of data storage is low, but SSD and additional RAM are also fully used if available.

## Parallel Processing on Multiple Cores

Large queries are parallelized naturally, taking all the necessary resources available on the current server.

## Distributed Processing on Multiple Servers

Almost none of the columnar DBMSs mentioned above have support for distributed query processing.

In ClickHouse, data can reside on different shards. Each shard can be a group of replicas used for fault tolerance. All shards are used to run a query in parallel, transparently for the user.

## SQL Support

ClickHouse supports a [declarative query language based on SQL](#) that is identical to the ANSI SQL standard in [many cases](#).

Supported queries include [GROUP BY](#), [ORDER BY](#), subqueries in [FROM](#), [JOIN](#) clause, [IN operator](#), [window functions](#) and scalar subqueries.

Correlated (dependent) subqueries are not supported at the time of writing but might become available in the future.

# Vector Computation Engine

Data is not only stored by columns but is processed by vectors (parts of columns), which allows achieving high CPU efficiency.

## Real-time Data Updates

ClickHouse supports tables with a primary key. To quickly perform queries on the range of the primary key, the data is sorted incrementally using the merge tree. Due to this, data can continually be added to the table. No locks are taken when new data is ingested.

## Primary Index

Having a data physically sorted by primary key makes it possible to extract data for its specific values or value ranges with low latency, less than a few dozen milliseconds.

## Secondary Indexes

Unlike other database management systems, secondary indexes in ClickHouse does not point to specific rows or row ranges. Instead, they allow the database to know in advance that all rows in some data parts wouldn't match the query filtering conditions and do not read them at all, thus they are called [data skipping indexes](#).

## Suitable for Online Queries

Most OLAP database management systems do not aim for online queries with sub-second latencies. In alternative systems, report building time of tens of seconds or even minutes is often considered acceptable. Sometimes it takes even more which forces to prepare reports offline (in advance or by responding with "come back later").

In ClickHouse low latency means that queries can be processed without delay and without trying to prepare an answer in advance, right at the same moment while the user interface page is loading. In other words, online.

## Support for Approximated Calculations

ClickHouse provides various ways to trade accuracy for performance:

1. Aggregate functions for approximated calculation of the number of distinct values, medians, and quantiles.
2. Running a query based on a part (sample) of data and getting an approximated result. In this case, proportionally less data is retrieved from the disk.
3. Running an aggregation for a limited number of random keys, instead of for all keys. Under certain conditions for key distribution in the data, this provides a reasonably accurate result while using fewer resources.

## Adaptive Join Algorithm

ClickHouse adaptively chooses how to [JOIN](#) multiple tables, by preferring hash-join algorithm and falling back to the merge-join algorithm if there's more than one large table.

## Data Replication and Data Integrity Support

ClickHouse uses asynchronous multi-master replication. After being written to any available replica, all the remaining replicas retrieve their copy in the background. The system maintains identical data on different replicas. Recovery after most failures is performed automatically, or semi-automatically in complex cases.

For more information, see the section [Data replication](#).

# Role-Based Access Control

ClickHouse implements user account management using SQL queries and allows for [role-based access control configuration](#) similar to what can be found in ANSI SQL standard and popular relational database management systems.

## Features that Can Be Considered Disadvantages

1. No full-fledged transactions.
2. Lack of ability to modify or delete already inserted data with a high rate and low latency. There are batch deletes and updates available to clean up or modify data, for example, to comply with [GDPR](#).
3. The sparse index makes ClickHouse not so efficient for point queries retrieving single rows by their keys.

---

## Performance

According to internal testing results at Yandex, ClickHouse shows the best performance (both the highest throughput for long queries and the lowest latency on short queries) for comparable operating scenarios among systems of its class that were available for testing. You can view the test results on a [separate page](#).

Numerous independent benchmarks came to similar conclusions. They are not difficult to find using an internet search, or you can see [our small collection of related links](#).

## Throughput for a Single Large Query

Throughput can be measured in rows per second or megabytes per second. If the data is placed in the page cache, a query that is not too complex is processed on modern hardware at a speed of approximately 2-10 GB/s of uncompressed data on a single server (for the most straightforward cases, the speed may reach 30 GB/s). If data is not placed in the page cache, the speed depends on the disk subsystem and the data compression rate. For example, if the disk subsystem allows reading data at 400 MB/s, and the data compression rate is 3, the speed is expected to be around 1.2 GB/s. To get the speed in rows per second, divide the speed in bytes per second by the total size of the columns used in the query. For example, if 10 bytes of columns are extracted, the speed is expected to be around 100-200 million rows per second.

The processing speed increases almost linearly for distributed processing, but only if the number of rows resulting from aggregation or sorting is not too large.

## Latency When Processing Short Queries

If a query uses a primary key and does not select too many columns and rows to process (hundreds of thousands), you can expect less than 50 milliseconds of latency (single digits of milliseconds in the best case) if data is placed in the page cache. Otherwise, latency is mostly dominated by the number of seeks. If you use rotating disk drives, for a system that is not overloaded, the latency can be estimated with this formula: `seek time (10 ms) * count of columns queried * count of data parts`

## Throughput When Processing a Large Quantity of Short Queries

Under the same conditions, ClickHouse can handle several hundred queries per second on a single server (up to several thousand in the best case). Since this scenario is not typical for analytical DBMSs, we recommend expecting a maximum of 100 queries per second.

## Performance When Inserting Data

We recommend inserting data in packets of at least 1000 rows, or no more than a single request per second. When inserting to a MergeTree table from a tab-separated dump, the insertion speed can be from 50 to 200 MB/s. If the inserted rows are around 1 KB in size, the speed will be from 50,000 to 200,000 rows per second. If the rows are small, the performance can be higher in rows per second (on Banner System data -> 500,000 rows per second; on Graphite data -> 1,000,000 rows per second). To improve performance, you can make multiple INSERT queries in parallel, which scales linearly.

## ClickHouse History

ClickHouse has been developed initially to power [Yandex.Metrica, the second largest web analytics platform in the world](#), and continues to be the core component of this system. With more than 13 trillion records in the database and more than 20 billion events daily, ClickHouse allows generating custom reports on the fly directly from non-aggregated data. This article briefly covers the goals of ClickHouse in the early stages of its development.

Yandex.Metrica builds customized reports on the fly based on hits and sessions, with arbitrary segments defined by the user. Doing so often requires building complex aggregates, such as the number of unique users. New data for building a report arrives in real-time.

As of April 2014, Yandex.Metrica was tracking about 12 billion events (page views and clicks) daily. All these events must be stored to build custom reports. A single query may require scanning millions of rows within a few hundred milliseconds, or hundreds of millions of rows in just a few seconds.

## Usage in Yandex.Metrica and Other Yandex Services

ClickHouse serves multiple purposes in Yandex.Metrica.

Its main task is to build reports in online mode using non-aggregated data. It uses a cluster of 374 servers, which store over 20.3 trillion rows in the database. The volume of compressed data is about 2 PB, without accounting for duplicates and replicas. The volume of uncompressed data (in TSV format) would be approximately 17 PB.

ClickHouse also plays a key role in the following processes:

- Storing data for Session Replay from Yandex.Metrica.
- Processing intermediate data.
- Building global reports with Analytics.
- Running queries for debugging the Yandex.Metrica engine.
- Analyzing logs from the API and the user interface.

Nowadays, there are multiple dozen ClickHouse installations in other Yandex services and departments: search verticals, e-commerce, advertisement, business analytics, mobile development, personal services, and others.

## Aggregated and Non-aggregated Data

There is a widespread opinion that to calculate statistics effectively, you must aggregate data since this reduces the volume of data.

But data aggregation comes with a lot of limitations:

- You must have a pre-defined list of required reports.
- The user can't make custom reports.

- When aggregating over a large number of distinct keys, the data volume is barely reduced, so aggregation is useless.
- For a large number of reports, there are too many aggregation variations (combinatorial explosion).
- When aggregating keys with high cardinality (such as URLs), the volume of data is not reduced by much (less than twofold).
- For this reason, the volume of data with aggregation might grow instead of shrink.
- Users do not view all the reports we generate for them. A large portion of those calculations is useless.
- The logical integrity of data may be violated for various aggregations.

If we do not aggregate anything and work with non-aggregated data, this might reduce the volume of calculations.

However, with aggregation, a significant part of the work is taken offline and completed relatively calmly. In contrast, online calculations require calculating as fast as possible, since the user is waiting for the result.

Yandex.Metrica has a specialized system for aggregating data called Metrage, which was used for the majority of reports.

Starting in 2009, Yandex.Metrica also used a specialized OLAP database for non-aggregated data called OLAPServer, which was previously used for the report builder.

OLAPServer worked well for non-aggregated data, but it had many restrictions that did not allow it to be used for all reports as desired. These included the lack of support for data types (only numbers), and the inability to incrementally update data in real-time (it could only be done by rewriting data daily).

OLAPServer is not a DBMS, but a specialized DB.

The initial goal for ClickHouse was to remove the limitations of OLAPServer and solve the problem of working with non-aggregated data for all reports, but over the years, it has grown into a general-purpose database management system suitable for a wide range of analytical tasks.

## ClickHouse Adopters

### Disclaimer

The following list of companies using ClickHouse and their success stories is assembled from public sources, thus might differ from current reality. We'd appreciate it if you share the story of adopting ClickHouse in your company and **add it to the list**, but please make sure you won't have any NDA issues by doing so. Providing updates with publications from other companies is also useful.

Company	Industry	Usecase	Cluster Size	(Un)Compressed Data Size* <u>(of single replica)</u>	Ref
2gis	Maps	Monitoring	—	—	Talk i July 2
Admiral	Martech	Engagement Management	—	—	Webin June 1
AdScribe	Ads	TV Analytics	—	—	A quick

Company	Industry	Usecase	Cluster Size	(Un)Compressed Data Size (of single replica)	Ref
Ahrefs	SEO	Analytics	—	—	Job listing
Alibaba Cloud	Cloud	Managed Service	—	—	Official
Alibaba Cloud	Cloud	E-MapReduce	—	—	Official
Aloha Browser	Mobile App	Browser backend	—	—	Slides   May 2
Altinity	Cloud, SaaS	Main product	—	—	Official
Amadeus	Travel	Analytics	—	—	Press April
ApiRoad	API marketplace	Analytics	—	—	Blog   2018
Appsflyer	Mobile analytics	Main product	—	—	Talk i July 2
ArenaData	Data Platform	Main product	—	—	Slides   December
Argedor	ClickHouse support	—	—	—	Official
Avito	Classifieds	Monitoring	—	—	Meeting
Badoo	Dating	Timeseries	—	1.6 mln events/sec (2018)	Slides   December
Beeline	Telecom	Data Platform	—	—	Blog   2021
Benocs	Network Telemetry and Analytics	Main Product	—	—	Slides   October
BIGO	Video	Computing Platform	—	—	Blog   August
BiliBili	Video sharing	—	—	—	Blog   2021
Bloomberg	Finance, Media	Monitoring	—	—	Job offer   September slides

Company	Industry	Usecase	Cluster Size	(Un)Compressed Data Size (of single replica)	Ref
Bloxy	Blockchain	Analytics	—	—	Slides   August
Bytedance	Social platforms	—	—	—	The C   Meeti   Octo
CardsMobile	Finance	Analytics	—	—	VC.ru
CARTO	Business Intelligence	Geo analytics	—	—	Geos   proce   Clickt
CERN	Research	Experiment	—	—	Press   2012
Checkly	Software Development	Analytics	—	—	Twee   2021
ChelPipe Group	Analytics	—	—	—	Blog   2021
Cisco	Networking	Traffic analysis	—	—	Lighti   Octo
Citadel Securities	Finance	—	—	—	Contr   Marcl
Citymobil	Taxi	Analytics	—	—	Blog   Russi   2020
Cloudflare	CDN	Traffic analysis	36 servers	—	Blog   2017   Marcl
Comcast	Media	CDN Traffic Analysis	—	—	Apac   Talk
ContentSquare	Web analytics	Main product	—	—	Blog   Frenc   2018
Corunet	Analytics	Main product	—	—	Slides   April
CreditX 慾信	Finance AI	Analysis	—	—	Slides   Nove

Company	Industry	Usecase	Cluster Size	(Un)Compressed Data Size (of single replica)	Ref
Crazypanda	Games		—	—	Live session, Clickstream analysis, Meetups
Criteo	Retail	Main product	—	—	Slides, Oct 2018
Cryptology	Digital Assets Trading Platform	—	—	—	Job ad, March 2019
Dataliance for China Telecom	Telecom	Analytics	—	—	Slides, January 2019
Deutsche Bank	Finance	BI Analytics	—	—	Slides, Oct 2018
Deepplay	Gaming Analytics	—	—	—	Job ad, 2020
Diva-e	Digital consulting	Main Product	—	—	Slides, September 2018
Ecommpay	Payment Processing	Logs	—	—	Videos
Ecwid	E-commerce SaaS	Metrics, Logging	—	—	Slides, April 2019
eBay	E-commerce	Logs, Metrics and Events	—	—	Official, Sep 2018
Exness	Trading	Metrics, Logging	—	—	Talk in May 2019
EventBunker.io	Serverless Data Processing	—	—	—	Tweets
FastNetMon	DDoS Protection	Main Product		—	Official
Flipkart	e-Commerce	—	—	—	Talk in July 2019
FunCorp	Games		—	14 bn records/day as of Jan 2021	Article
Geniee	Ad network	Main product	—	—	Blog   Japan 2017

Company	Industry	Usecase	Cluster Size	(Un)Compressed Data Size (of single replica)	Ref
Genotek	Bioinformatics	Main product	—	—	<a href="#">Video 2020</a>
Gigapipe	Managed ClickHouse	Main product	—	—	<a href="#">Official</a>
Glaber	Monitoring	Main product	—	—	<a href="#">Webs</a>
GraphCDN	CDN	Traffic Analytics	—	—	<a href="#">Blog   English 2021</a>
HUYA	Video Streaming	Analytics	—	—	<a href="#">Slides Octo</a>
Hydrolix	Cloud data platform	Main product	—	—	<a href="#">Docu</a>
ICA	FinTech	Risk Management	—	—	<a href="#">Blog   English</a>
Idealista	Real Estate	Analytics	—	—	<a href="#">Blog   English</a>
Infobaleen	AI markting tool	Analytics	—	—	<a href="#">Official</a>
Infovista	Networks	Analytics	—	—	<a href="#">Slides Octo</a>
InnoGames	Games	Metrics, Logging	—	—	<a href="#">Slides Septe</a>
Instabug	APM Platform	Main product	—	—	<a href="#">A quick Found</a>
Instana	APM Platform	Main product	—	—	<a href="#">Twitter</a>
Integros	Platform for video services	Analytics	—	—	<a href="#">Slides May 2</a>
Ippon Technologies	Technology Consulting	—	—	—	<a href="#">Talk i July 2</a>
Ivi	Online Cinema	Analytics, Monitoring	—	—	<a href="#">Article Jan 21</a>
Jinshuju 金数据	BI Analytics	Main product	—	—	<a href="#">Slides Octo</a>

Company	Industry	Usecase	Cluster Size	(Un)Compressed Data Size (of single replica)	Refere
kakaocorp	Internet company	—	—	—	if(kak confe
Kodiak Data	Clouds	Main product	—	—	Slides April
Kontur	Software Development	Metrics	—	—	Talk i Nove
Kuaishou	Video	—	—	—	ClickF Meeti 2018
KGK Global	Vehicle monitoring	—	—	—	Press 2021
Lawrence Berkeley National Laboratory	Research	Traffic analysis	1 server	11.8 TiB	Slides April
LifeStreet	Ad network	Main product	75 servers (3 replicas)	5.27 PiB	Blog   Russi 2017
Mail.ru Cloud Solutions	Cloud services	Main product	—	—	Articl
MAXILECT	Ad Tech, Blockchain, ML, AI	—	—	—	Job a 2021
Marilyn	Advertising	Statistics	—	—	Talk i June :
Mello	Marketing	Analytics	1 server	—	Articl
MessageBird	Telecommunications	Statistics	—	—	Slides Nove
Microsoft	Web Analytics	Clarity (Main Product)	—	—	A que GitHub
MindsDB	Machine Learning	Main Product	—	—	Offici
MUX	Online Video	Video Analytics	—	—	Talk i Augu

Company	Industry	Usecase	Cluster Size	(Un)Compressed Data Size (of single replica)	Ref
MGID	Ad network	Web-analytics	—	—	Blog   Russia 2020
Netskope	Network Security	—	—	—	Job ad March
NIC Labs	Network Monitoring	RaTA-DNS	—	—	Blog   2021
NOC Project	Network Monitoring	Analytics	Main Product	—	Official
Noction	Network Technology	Main Product	—	—	Official
Nuna Inc.	Health Data Analytics	—	—	—	Talk i July 2
Ok.ru	Social Network	—	72 servers	810 TB compressed, 50bn rows/day, 1.5 TB/day	Smart conf 2021
Omnicomm	Transportation Monitoring	—	—	—	Facebook Oct 2
OneAPM	Monitorings and Data Analysis	Main product	—	—	Slides Oct 2
OZON	E-commerce	—	—	—	Official
Panelbear	Analytics	Monitoring and Analytics	—	—	Tech Nove
Percent 百分点	Analytics	Main Product	—	—	Slides June 1
Percona	Performance analysis	Percona Monitoring and Management	—	—	Official Mar 2
Plausible	Analytics	Main Product	—	—	Blog   2020
PostHog	Product Analytics	Main Product	—	—	Release 2020

Company	Industry	Usecase	Cluster Size	(Un)Compressed Data Size (of single replica)	Ref
Postmates	Delivery	—	—	—	Talk i July 2
Pragma Innovation	Telemetry and Big Data Analysis	Main product	—	—	Slides Octok
PRANA	Industrial predictive analytics	Main product	—	—	News Feb 2
QINGCLOUD	Cloud services	Main product	—	—	Slides Octok
Qrator	DDoS protection	Main product	—	—	Blog 1 2019
Raiffeisenbank	Banking	Analytics	—	—	Lecture Decem
Rambler	Internet services	Analytics	—	—	Talk i April
Replica	Urban Planning	Analytics	—	—	Job ad
Retell	Speech synthesis	Analytics	—	—	Blog 7 August
Rollbar	Software Development	Main Product	—	—	Official
Rspamd	Antispam	Analytics	—	—	Official
RuSIEM	SIEM	Main Product	—	—	Official
S7 Airlines	Airlines	Metrics, Logging	—	—	Talk i March
Sber	Banking, Fintech, Retail, Cloud, Media	—	—	—	Job ad March
scireum GmbH	e-Commerce	Main product	—	—	Talk i Febru

Company	Industry	Usecase	Cluster Size	(Un)Compressed Data Size (of single replica)	Ref
Segment	Data processing	Main product	9 * i3en.3xlarge nodes 7.5TB NVME SSDs, 96GB Memory, 12 vCPUs	—	Slides   Slides
sembot.io	Shopping Ads	—	—	—	A con Linke
SEMrush	Marketing	Main product	—	—	Slides   Augus
Sentry	Software Development	Main product	—	—	Blog   English
seo.do	Analytics	Main product	—	—	Slides   Nove
SGK	Government Social Security	Analytics	—	—	Slides   Nove
SigNoz	Observability Platform	Main Product	—	—	Source
Sina	News	—	—	—	Slides   Octob
Sipfront	Software Development	Analytics	—	—	Twee 2021
SMI2	News	Analytics	—	—	Blog   Russi Nove
Spark New Zealand	Telecommunications	Security Operations	—	—	Blog   2020
Splitbee	Analytics	Main Product	—	—	Blog   2021
Splunk	Business Analytics	Main product	—	—	Slides   Janua
Spotify	Music	Experimentation	—	—	Slides

Company	Industry	Usecase	Cluster Size	(Un)Compressed Data Size (of single replica)	Ref
Staffcop	Information Security	Main Product	—	—	Official Docu
Suning	E-Commerce	User behaviour analytics	—	—	Blog ↗
Teralytics	Mobility	Analytics	—	—	Tech ↗
Tencent	Big Data	Data processing	—	—	Slides Octok ↗
Tencent	Messaging	Logging	—	—	Talk i Nove ↗
Tencent Music Entertainment (TME)	BigData	Data processing	—	—	Blog i June ↗
Tesla	Electric vehicle and clean energy company	—	—	—	Vacai descr 2021 ↗
Timeflow	Software	Analytics	—	—	Blog ↗
Tinybird	Real-time Data Products	Data processing	—	—	Official ↗
Traffic Stars	AD network	—	300 servers in Europe/US	1.8 PiB, 700 000 insert rps (as of 2021)	Slides May ↗
Uber	Taxi	Logging	—	—	Slides 2020 ↗
UTMSTAT	Analytics	Main product	—	—	Blog i 2020 ↗
VKontakte	Social Network	Statistics, Logging	—	—	Slides Aug ↗
VMware	Cloud	VeloCloud, SDN	—	—	Product docur ↗
Walmart Labs	Internet, Retail	—	—	—	Talk i July 2 ↗
Wargaming	Games	—	—	—	Intern ↗

Company	Industry	Usecase	Cluster Size	(Un)Compressed Data Size (of single replica)	Ref
Wildberries	E-commerce		—	—	Official presentation
Wisebits	IT Solutions	Analytics	—	—	Slides from May 2021
Workato	Automation Software	—	—	—	Talk in July 2021
Xenoss	Marketing, Advertising	—	—	—	Instagram post from 2021
Xiaoxin Tech	Education	Common purpose	—	—	Slides from November 2021
Ximalaya	Audio sharing	OLAP	—	—	Slides from November 2021
Yandex Cloud	Public Cloud	Main product	—	—	Talk in December 2021
Yandex DataLens	Business Intelligence	Main product	—	—	Slides from December 2021
Yandex Market	e-Commerce	Metrics, Logging	—	—	Talk in January 2022
Yandex Metrica	Web analytics	Main product	630 servers in one cluster, 360 servers in another cluster, 1862 servers in one department	133 PiB / 8.31 PiB / 120 trillion records	Slides from 2020
Yotascale	Cloud	Data pipeline	—	2 bn records/day	Link (Account)
Zagrava Trading	—	—	—	—	Job offer from 2021
ЦВТ	Software Development	Metrics, Logging	—	—	Blog post from 2019
МКБ	Bank	Web-system monitoring	—	—	Slides from September 2021

Company	Industry	Usecase	Cluster Size	(Un)Compressed Data Size (of single replica)	Ref...
ЦФТ	Banking, Financial products, Payments	—	—	—	Meeting April
Цифровой Рабочий	Industrial IoT, Analytics	—	—	—	Blog   Russia 2021
ООО «МПЗ Богородский»	Agriculture	—	—	—	Article November
ДомКлик	Real Estate	—	—	—	Article October
DeepL	Machine Learning	—	—	—	Video 2021

## Getting Started

If you are new to ClickHouse and want to get a hands-on feeling of its performance, first of all, you need to go through the [installation process](#). After that you can:

- [Go through detailed tutorial](#)
- [Experiment with example datasets](#)

## Example Datasets

This section describes how to obtain example datasets and import them into ClickHouse. For some datasets example queries are also available.

The list of documented datasets:

- [GitHub Events](#)
- [Anonymized Yandex.Metrica Dataset](#)
- [Recipes](#)
- [Star Schema Benchmark](#)
- [WikiStat](#)
- [Terabyte of Click Logs from Criteo](#)
- [AMPLab Big Data Benchmark](#)
- [Brown University Benchmark](#)
- [New York Taxi Data](#)
- [OpenSky](#)

- [UK Property Price Paid](#)
  - [Cell Towers](#)
  - [What's on the Menu?](#)
  - [OnTime](#)
- 

## GitHub Events Dataset

Dataset contains all events on GitHub from 2011 to Dec 6 2020, the size is 3.1 billion records. Download size is 75 GB and it will require up to 200 GB space on disk if stored in a table with lz4 compression.

Full dataset description, insights, download instruction and interactive queries are posted [here](#).

---

## Anonymized Yandex.Metrica Data

Dataset consists of two tables containing anonymized data about hits (`hits_v1`) and visits (`visits_v1`) of Yandex.Metrica. You can read more about Yandex.Metrica in [ClickHouse history](#) section.

The dataset consists of two tables, either of them can be downloaded as a compressed `tsv.xz` file or as prepared partitions. In addition to that, an extended version of the `hits` table containing 100 million rows is available as TSV at [https://datasets.clickhouse.com/hits/tsv/hits\\_100m\\_obfuscated\\_v1.tsv.xz](https://datasets.clickhouse.com/hits/tsv/hits_100m_obfuscated_v1.tsv.xz) and as prepared partitions at [https://datasets.clickhouse.com/hits/partitions/hits\\_100m\\_obfuscated\\_v1.tar.xz](https://datasets.clickhouse.com/hits/partitions/hits_100m_obfuscated_v1.tar.xz).

## Obtaining Tables from Prepared Partitions

Download and import hits table:

```
curl -O https://datasets.clickhouse.com/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"
```

Download and import visits:

```
curl -O https://datasets.clickhouse.com/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"
```

## Obtaining Tables from Compressed TSV File

Download and import hits from compressed TSV file:

```

curl https://datasets.clickhouse.com/hits/tsv/hits_v1.tsv.xz | unxz --threads=`nproc` > hits_v1.tsv
## Validate the checksum
md5sum hits_v1.tsv
## Checksum should be equal to: f3631b6295bf06989c1437491f7592cb
## now create table
clickhouse-client --query "CREATE DATABASE IF NOT EXISTS datasets"
## for hits_v1
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 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, 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"
## for hits_100m_obfuscated
clickhouse-client --query="CREATE TABLE hits_100m_obfuscated (WatchID UInt64, JavaEnable UInt8, Title String,
GoodEvent Int16, EventTime DateTime, EventDate Date, CounterID UInt32, ClientIP UInt32, RegionID UInt32, UserID
UInt64, CounterClass Int8, OS UInt8, UserAgent UInt8, URL String, Referer String, Refresh UInt8, RefererCategoryID
UInt16, RefererRegionID UInt32, URLCategoryID UInt16, URLRegionID 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,
OriginalURL String, HID UInt32, IsOldCounter UInt8, IsEvent UInt8, IsParameter UInt8, DontCountHits UInt8, WithHash
UInt8, HitColor FixedString(1), LocalEventTime DateTime, Age UInt8, Sex UInt8, Income UInt8, Interests UInt16,
Robotness UInt8, RemoteIP UInt32, WindowName Int32, OpenerName Int32, HistoryLength Int16, BrowserLanguage
FixedString(2), BrowserCountry FixedString(2), SocialNetwork String, SocialAction String, HTTPError UInt16,
SendTiming UInt32, DNSTiming UInt32, ConnectTiming UInt32, ResponseStartTiming UInt32, ResponseEndTiming
UInt32, FetchTiming UInt32, SocialSourceNetworkID UInt8, SocialSourcePage String, ParamPrice Int64, ParamOrderID
String, ParamCurrency FixedString(3), ParamCurrencyID UInt16, 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) 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"

```

Download and import visits from compressed tsv-file:

```

curl https://datasets.clickhouse.com/visits/tsv/visits_v1.tsv.xz | unxz --threads=`nproc` > visits_v1.tsv
## Validate the checksum
md5sum visits_v1.tsv
## Checksum should be equal to: 6daf1a0f24e59e3fc2d0fed85601de6
## 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 UInt32, Hits Int32, IsBounce UInt8, Referrer String, StartURL String, RefererDomain String, StartURLDomain String, EndURL String, LinkURL String, IsDownload UInt8, TraficSourceID Int8, SearchEnginID UInt16, SearchPhrase String, AdvEnginID 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, 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 UInt8, SearchEngineID 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, 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"

```

## Example Queries

[ClickHouse tutorial](#) is based on Yandex.Metrica dataset and the recommended way to get started with this dataset is to just go through tutorial.

Additional examples of queries to these tables can be found among [stateful tests](#) of ClickHouse (they are named `test.hits` and `test.visits` there).

## Recipes Dataset

RecipeNLG dataset is available for download [here](#). It contains 2.2 million recipes. The size is slightly less than 1 GB.

## Download and Unpack the Dataset

1. Go to the download page <https://recipenlg.cs.put.poznan.pl/dataset>.
2. Accept Terms and Conditions and download zip file.
3. Unpack the zip file with `unzip`. You will get the `full_dataset.csv` file.

## Create a Table

Run clickhouse-client and execute the following CREATE query:

```
CREATE TABLE recipes
(
    title String,
    ingredients Array(String),
    directions Array(String),
    link String,
    source LowCardinality(String),
    NER Array(String)
) ENGINE = MergeTree ORDER BY title;
```

## Insert the Data

Run the following command:

```
clickhouse-client --query "
    INSERT INTO recipes
    SELECT
        title,
        JSONExtract(ingredients, 'Array(String)'),
        JSONExtract(directions, 'Array(String)'),
        link,
        source,
        JSONExtract(NER, 'Array(String)')
    FROM input('num UInt32, title String, ingredients String, directions String, link String, source LowCardinality(String),
    NER String')
    FORMAT CSVWithNames
" --input_format_with_names_use_header 0 --format_csv_allow_single_quote 0 --input_format_allow_errors_num 10 <
full_dataset.csv
```

This is a showcase how to parse custom CSV, as it requires multiple tunes.

Explanation:

- The dataset is in CSV format, but it requires some preprocessing on insertion; we use table function `input` to perform preprocessing;
- The structure of CSV file is specified in the argument of the table function `input`;
- The field num (row number) is unneeded - we parse it from file and ignore;
- We use `FORMAT CSVWithNames` but the header in CSV will be ignored (by command line parameter `--input_format_with_names_use_header 0`), because the header does not contain the name for the first field;
- File is using only double quotes to enclose CSV strings; some strings are not enclosed in double quotes, and single quote must not be parsed as the string enclosing - that's why we also add the `--format_csv_allow_single_quote 0` parameter;
- Some strings from CSV cannot parse, because they contain `\M` sequence at the beginning of the value; the only value starting with backslash in CSV can be `\N` that is parsed as SQL NULL. We add `--input_format_allow_errors_num 10` parameter and up to ten malformed records can be skipped;
- There are arrays for ingredients, directions and NER fields; these arrays are represented in unusual form: they are serialized into string as JSON and then placed in CSV - we parse them as String and then use `JSONExtract` function to transform it to Array.

## Validate the Inserted Data

By checking the row count:

Query:

```
SELECT count() FROM recipes;
```

Result:

```
count()  
2231141
```

## Example Queries

### Top Components by the Number of Recipes:

In this example we learn how to use [arrayJoin](#) function to expand an array into a set of rows.

Query:

```
SELECT  
    arrayJoin(NER) AS k,  
    count() AS c  
FROM recipes  
GROUP BY k  
ORDER BY c DESC  
LIMIT 50
```

Result:

k	c
salt	890741
sugar	620027
butter	493823
flour	466110
eggs	401276
onion	372469
garlic	358364
milk	346769
water	326092
vanilla	270381
olive oil	197877
pepper	179305
brown sugar	174447
tomatoes	163933
egg	160507
baking powder	148277
lemon juice	146414
Salt	122557
cinnamon	117927
sour cream	116682
cream cheese	114423
margarine	112742
celery	112676
baking soda	110690
parsley	102151
chicken	101505
onions	98903
vegetable oil	91395
oil	85600
mayonnaise	84822
pecans	79741
nuts	78471
potatoes	75820
carrots	75458
pineapple	74345
soy sauce	70355
black pepper	69064
thyme	68429
mustard	65948
chicken broth	65112
bacon	64956
honey	64626
oregano	64077
ground beef	64068
unsalted butter	63848
mushrooms	61465
Worcestershire sauce	59328
cornstarch	58476
green pepper	58388
Cheddar cheese	58354

50 rows in set. Elapsed: 0.112 sec. Processed 2.23 million rows, 361.57 MB (19.99 million rows/s., 3.24 GB/s.)

## The Most Complex Recipes with Strawberry

```

SELECT
    title,
    length(NER),
    length(directions)
FROM recipes
WHERE has(NER, 'strawberry')
ORDER BY length(directions) DESC
LIMIT 10

```

Result:

title			length(NER)	length(directions)
Chocolate-Strawberry-Orange Wedding Cake		24	126	
Strawberry Cream Cheese Crumble Tart		19	47	
Charlotte-Style Ice Cream	11		45	
Sinfully Good a Million Layers Chocolate Layer Cake, With Strawb		31		45
Sweetened Berries With Elderflower Sherbet	24		44	
Chocolate-Strawberry Mousse Cake	15		42	
Rhubarb Charlotte with Strawberries and Rum	20		42	
Chef Joey's Strawberry Vanilla Tart	7		37	
Old-Fashioned Ice Cream Sundae Cake	17		37	
Watermelon Cake	16	36		

10 rows in set. Elapsed: 0.215 sec. Processed 2.23 million rows, 1.48 GB (10.35 million rows/s., 6.86 GB/s.)

In this example, we involve **has** function to filter by array elements and sort by the number of directions.

There is a wedding cake that requires the whole 126 steps to produce! Show that directions:

Query:

```
SELECT arrayJoin(directions)
FROM recipes
WHERE title = 'Chocolate-Strawberry-Orange Wedding Cake'
```

Result:

arrayJoin(directions)

Position 1 rack in center and 1 rack in bottom third of oven and preheat to 350F.

Butter one 5-inch-diameter cake pan with 2-inch-high sides, one 8-inch-diameter cake pan with 2-inch-high sides and one 12-inch-diameter cake pan with 2-inch-high sides.

Dust pans with flour; line bottoms with parchment.

Combine 1/3 cup orange juice and 2 ounces unsweetened chocolate in heavy small saucepan.

Stir mixture over medium-low heat until chocolate melts.

Remove from heat.

Gradually mix in 1 2/3 cups orange juice.

Sift 3 cups flour, 2/3 cup cocoa, 2 teaspoons baking soda, 1 teaspoon salt and 1/2 teaspoon baking powder into medium bowl.

using electric mixer, beat 1 cup (2 sticks) butter and 3 cups sugar in large bowl until blended (mixture will look grainy).

Add 4 eggs, 1 at a time, beating to blend after each.

Beat in 1 tablespoon orange peel and 1 tablespoon vanilla extract.

Add dry ingredients alternately with orange juice mixture in 3 additions each, beating well after each addition.

Mix in 1 cup chocolate chips.

Transfer 1 cup plus 2 tablespoons batter to prepared 5-inch pan, 3 cups batter to prepared 8-inch pan and remaining batter (about 6 cups) to 12-inch pan.

Place 5-inch and 8-inch pans on center rack of oven.

Place 12-inch pan on lower rack of oven.

Bake cakes until tester inserted into center comes out clean, about 35 minutes.

Transfer cakes in pans to racks and cool completely.

Mark 4-inch diameter circle on one 6-inch-diameter cardboard cake round.

Cut out marked circle.

Mark 7-inch-diameter circle on one 8-inch-diameter cardboard cake round.

Cut out marked circle.

Mark 11-inch-diameter circle on one 12-inch-diameter cardboard cake round.

Cut out marked circle.

Cut around sides of 5-inch-cake to loosen.

Place 4-inch cardboard over pan.

Hold cardboard and pan together; turn cake out onto cardboard.

Peel off parchment. Wrap cakes on its cardboard in foil.

Repeat turning out, peeling off parchment and wrapping cakes in foil, using 7-inch cardboard for 8-inch cake and 11-inch cardboard for 12-inch cake.

Using remaining ingredients, make 1 more batch of cake batter and bake 3 more cake layers as described above.

Cool cakes in pans.

Cover cakes in pans tightly with foil.

(Can be prepared ahead.)

Let stand at room temperature up to 1 day or double-wrap all cake layers and freeze up to 1 week.

Bring cake layers to room temperature before using.)

Place first 12-inch cake on its cardboard on work surface.

Spread 2 3/4 cups ganache over top of cake and all the way to edge.

Spread 2/3 cup jam over ganache, leaving 1/2-inch chocolate border at edge.

Drop 1 3/4 cups white chocolate frosting by spoonfuls over jam.

Gently spread frosting over jam, leaving 1/2-inch chocolate border at edge.

Rub some cocoa powder over second 12-inch cardboard.

Cut around sides of second 12-inch cake to loosen.

Place cardboard, cocoa side down, over pan.

Turn cake out onto cardboard.

Peel off parchment.

Carefully slide cake off cardboard and onto filling on first 12-inch cake.

Refrigerate.

Place first 8-inch cake on its cardboard on work surface.

Spread 1 cup ganache over top all the way to edge.

Spread 1/4 cup jam over, leaving 1/2-inch chocolate border at edge.

Drop 1 cup white chocolate frosting by spoonfuls over jam.

Gently spread frosting over jam, leaving 1/2-inch chocolate border at edge.

Rub some cocoa over second 8-inch cardboard.

Cut around sides of second 8-inch cake to loosen.

Place cardboard, cocoa side down, over pan.

Turn cake out onto cardboard.

Peel off parchment.

Slide cake off cardboard and onto filling on first 8-inch cake.

Refrigerate.

Place first 5-inch cake on its cardboard on work surface.

Spread 1/2 cup ganache over top of cake and all the way to edge.

Spread 2 tablespoons jam over, leaving 1/2-inch chocolate border at edge.

Drop 1/3 cup white chocolate frosting by spoonfuls over jam.

Gently spread frosting over jam, leaving 1/2-inch chocolate border at edge.

Rub cocoa over second 6-inch cardboard.

Cut around sides of second 5-inch cake to loosen.

Place cardboard, cocoa side down, over pan.

Turn cake out onto cardboard.

Peel off parchment.

Slide cake off cardboard and onto filling on first 5-inch cake.

Chill all cakes 1 hour to set filling.

Place 12-inch tiered cake on its cardboard on revolving cake stand.

Spread 2 2/3 cups frosting over top and sides of cake as a first coat.

Refrigerate cake.

Place 8-inch tiered cake on its cardboard on cake stand.

Spread 1 1/4 cups frosting over top and sides of cake as a first coat.

Refrigerate cake.

Place 5-inch tiered cake on its cardboard on cake stand.

Spread 3/4 cup frosting over top and sides of cake as a first coat.

Refrigerate all cakes until first coats of frosting set, about 1 hour.

(Cakes can be made to this point up to 1 day ahead; cover and keep refrigerate.)

Prepare second batch of frosting, using remaining frosting ingredients and following directions for first batch.

Spoon 2 cups frosting into pastry bag fitted with small star tip.

Place 12-inch cake on its cardboard on large flat platter.

Place platter on cake stand.

Using icing spatula, spread 2 1/2 cups frosting over top and sides of cake; smooth top.

Using filled pastry bag, pipe decorative border around top edge of cake.

Refrigerate cake on platter.

Place 8-inch cake on its cardboard on cake stand.

Using icing spatula, spread 1 1/2 cups frosting over top and sides of cake; smooth top.

Using pastry bag, pipe decorative border around top edge of cake.

Refrigerate cake on its cardboard.

Place 5-inch cake on its cardboard on cake stand.

Using icing spatula, spread 3/4 cup frosting over top and sides of cake; smooth top.

Using pastry bag, pipe decorative border around top edge of cake, spooning more frosting into bag if necessary.

Refrigerate cake on its cardboard.

Keep all cakes refrigerated until frosting sets, about 2 hours.

(Can be prepared 2 days ahead.

Cover loosely; keep refrigerated.)

Place 12-inch cake on platter on work surface.

Press 1 wooden dowel straight down into and completely through center of cake.

Mark dowel 1/4 inch above top of frosting.

Remove dowel and cut with serrated knife at marked point.

Cut 4 more dowels to same length.

Press 1 cut dowel back into center of cake.

Press remaining 4 cut dowels into cake, positioning 3 1/2 inches inward from cake edges and spacing evenly.

Place 8-inch cake on its cardboard on work surface.

Press 1 dowel straight down into and completely through center of cake.

Mark dowel 1/4 inch above top of frosting.

Remove dowel and cut with serrated knife at marked point.

Cut 3 more dowels to same length

Cut 3 more dowels to same length.  
Press 1 cut dowel back into center of cake.

Press remaining 3 cut dowels into cake, positioning 2 1/2 inches inward from edges and spacing evenly.

Using large metal spatula as aid, place 8-inch cake on its cardboard atop dowels in 12-inch cake, centering carefully.

Gently place 5-inch cake on its cardboard atop dowels in 8-inch cake, centering carefully.

Using citrus stripper, cut long strips of orange peel from oranges.

Cut strips into long segments.

To make orange peel coils, wrap peel segment around handle of wooden spoon; gently slide peel off handle so that peel keeps coiled shape.

Garnish cake with orange peel coils, ivy or mint sprigs, and some berries.

(Assembled cake can be made up to 8 hours ahead.

Let stand at cool room temperature.)

Remove top and middle cake tiers.

Remove dowels from cakes.

Cut top and middle cakes into slices.

To cut 12-inch cake: Starting 3 inches inward from edge and inserting knife straight down, cut through from top to bottom to make 6-inch-diameter circle in center of cake.

Cut outer portion of cake into slices; cut inner portion into slices and serve with strawberries.

126 rows in set. Elapsed: 0.011 sec. Processed 8.19 thousand rows, 5.34 MB (737.75 thousand rows/s., 480.59 MB/s.)

## Online Playground

The dataset is also available in the [Online Playground](#).

## Star Schema Benchmark

Compiling dbgen:

```
$ git clone git@github.com:vadimtk/ssb-dbgen.git  
$ cd ssb-dbgen  
$ make
```

Generating data:

### Attention

With `-s 100` dbgen generates 600 million rows (67 GB), while while `-s 1000` it generates 6 billion rows (which takes a lot of time)

```
$ ./dbgen -s 1000 -T c  
$ ./dbgen -s 1000 -T I  
$ ./dbgen -s 1000 -T p  
$ ./dbgen -s 1000 -T s  
$ ./dbgen -s 1000 -T d
```

Creating tables in ClickHouse:

```

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;

```

Inserting data:

```

$ 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

```

Converting “star schema” to denormalized “flat schema”:

```

SET max_memory_usage = 200000000000;

CREATE TABLE lineorder_flat
ENGINE = MergeTree
PARTITION BY toYear(LO_ORDERDATE)
ORDER BY (LO_ORDERDATE, LO_ORDERKEY) AS
SELECT
    I.LO_ORDERKEY AS LO_ORDERKEY,
    I.LO_LINENUMBER AS LO_LINENUMBER,
    I.LO_CUSTKEY AS LO_CUSTKEY,
    I.LO_PARTKEY AS LO_PARTKEY,
    I.LO_SUPPKEY AS LO_SUPPKEY,
    I.LO_ORDERDATE AS LO_ORDERDATE,
    I.LO_ORDERPRIORITY AS LO_ORDERPRIORITY,
    I.LO_SHIPPRIORITY AS LO_SHIPPRIORITY,
    I.LO_QUANTITY AS LO_QUANTITY,
    I.LO_EXTENDEDPRICE AS LO_EXTENDEDPRICE,
    I.LO_ORDTOTALPRICE AS LO_ORDTOTALPRICE,
    I.LO_DISCOUNT AS LO_DISCOUNT,
    I.LO_REVENUE AS LO_REVENUE,
    I.LO_SUPPLYCOST AS LO_SUPPLYCOST,
    I.LO_TAX AS LO_TAX,
    I.LO_COMMITDATE AS LO_COMMITDATE,
    I.LO_SHIPMODE AS LO_SHIPMODE,
    c.C_NAME AS C_NAME,
    c.C_ADDRESS AS C_ADDRESS,
    c.C_CITY AS C_CITY,
    c.C_NATION AS C_NATION,
    c.C_REGION AS C_REGION,
    c.C_PHONE AS C_PHONE,
    c.C_MKTSEGMENT AS C_MKTSEGMENT,
    s.S_NAME AS S_NAME,
    s.S_ADDRESS AS S_ADDRESS,
    s.S_CITY AS S_CITY,
    s.S_NATION AS S_NATION,
    s.S_REGION AS S_REGION,
    s.S_PHONE AS S_PHONE,
    p.P_NAME AS P_NAME,
    p.P_MFGR AS P_MFGR,
    p.P_CATEGORY AS P_CATEGORY,
    p.P_BRAND AS P_BRAND,
    p.P_COLOR AS P_COLOR,
    p.P_TYPE AS P_TYPE,
    p.P_SIZE AS P_SIZE,
    p.P_CONTAINER AS P_CONTAINER
FROM lineorder AS I
INNER JOIN customer AS c ON c.C_CUSTKEY = I.LO_CUSTKEY
INNER JOIN supplier AS s ON s.S_SUPPKEY = I.LO_SUPPKEY
INNER JOIN part AS p ON p.P_PARTKEY = I.LO_PARTKEY;

```

Running the queries:

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 >= 'MFGR#2221' AND P_BRAND <= '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 ASC,
    C_NATION ASC;
```

#### 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 ASC,
    S_NATION ASC,
    P_CATEGORY ASC;
```

#### 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 ASC,
    S_CITY ASC,
    P_BRAND ASC;
```

## WikiStat

See: <http://dumps.wikimedia.org/other/pagecounts-raw/>

Creating a table:

```

CREATE TABLE wikistat
(
    date Date,
    time DateTime,
    project String,
    subproject String,
    path String,
    hits UInt64,
    size UInt64
) ENGINE = MergeTree(date, (path, time), 8192);

```

Loading data:

```

$ 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

```

## Terabyte of Click Logs from Criteo

Download the data from <http://labs.criteo.com/downloads/download-terabyte-click-logs/>

Create a table to import the log to:

```

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

```

Download the data:

```

$ 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-perftest01j --query="INSERT INTO criteo_log FORMAT TabSeparated"; done

```

Create a table for the converted data:

```

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)

```

Transform data from the raw log and put it in the second table:

```

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;

```

## AMPLab Big Data Benchmark

See <https://amplab.cs.berkeley.edu/benchmark/>

Sign up for a free account at <https://aws.amazon.com>. It requires a credit card, email, and phone number.  
Get a new access key at [https://console.aws.amazon.com/iam/home?nc2=h\\_m\\_sc#security\\_credential](https://console.aws.amazon.com/iam/home?nc2=h_m_sc#security_credential)

Run the following in the console:

```
$ 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 ..
```

Run the following ClickHouse queries:

```
CREATE TABLE rankings_tiny
(
    pageURL String,
    pageRank UInt32,
    avgDuration UInt32
) 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);
```

Go back to the console:

```
$ for i in tiny/rankings/*.deflate; do echo $i; zlib-flate -uncompress < $i | clickhouse-client --host=example-perftest01j --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-perftest01j --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-perftest01j --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-perftest01j --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-perftest01j --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-perftest01j --query="INSERT INTO uservisits_5nodes_on_single FORMAT CSV"; done
```

Queries for obtaining data samples:

```
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
```

## Brown University Benchmark

MgBench is a new analytical benchmark for machine-generated log data, [Andrew Crotty](#).

Download the data:

```
wget https://datasets.clickhouse.com/mgbench{1..3}.csv.xz
```

Unpack the data:

```
xz -v -d mgbench{1..3}.csv.xz
```

Create tables:

```

CREATE DATABASE mgbench;

CREATE TABLE mgbench.logs1 (
    log_time    DateTime,
    machine_name LowCardinality(String),
    machine_group LowCardinality(String),
    cpu_idle    Nullable(Float32),
    cpu_nice    Nullable(Float32),
    cpu_system  Nullable(Float32),
    cpu_user    Nullable(Float32),
    cpu_wio     Nullable(Float32),
    disk_free   Nullable(Float32),
    disk_total  Nullable(Float32),
    part_max_used Nullable(Float32),
    load_fifteen Nullable(Float32),
    load_five   Nullable(Float32),
    load_one    Nullable(Float32),
    mem_buffers Nullable(Float32),
    mem_cached   Nullable(Float32),
    mem_free    Nullable(Float32),
    mem_shared   Nullable(Float32),
    swap_free   Nullable(Float32),
    bytes_in    Nullable(Float32),
    bytes_out   Nullable(Float32)
)
ENGINE = MergeTree()
ORDER BY (machine_group, machine_name, log_time);

CREATE TABLE mgbench.logs2 (
    log_time    DateTime,
    client_ip   IPv4,
    request     String,
    status_code UInt16,
    object_size UInt64
)
ENGINE = MergeTree()
ORDER BY log_time;

CREATE TABLE mgbench.logs3 (
    log_time    DateTime64,
    device_id   FixedString(15),
    device_name LowCardinality(String),
    device_type LowCardinality(String),
    device_floor UInt8,
    event_type  LowCardinality(String),
    event_unit   FixedString(1),
    event_value  Nullable(Float32)
)
ENGINE = MergeTree()
ORDER BY (event_type, log_time);

```

Insert data:

```

clickhouse-client --query "INSERT INTO mgbench.logs1 FORMAT CSVWithNames" < mgbench1.csv
clickhouse-client --query "INSERT INTO mgbench.logs2 FORMAT CSVWithNames" < mgbench2.csv
clickhouse-client --query "INSERT INTO mgbench.logs3 FORMAT CSVWithNames" < mgbench3.csv

```

Run benchmark queries:

-- Q1.1: What is the CPU/network utilization for each web server since midnight?

```

SELECT machine_name,
       MIN(cpu) AS cpu_min,
       MAX(cpu) AS cpu_max,
       AVG(cpu) AS cpu_avg,
       MIN(net_in) AS net_in_min,
       MAX(net_in) AS net_in_max,
       AVG(net_in) AS net_in_avg,
       MIN(net_out) AS net_out_min

```

```

MIN(net_out) AS net_out_min,
MAX(net_out) AS net_out_max,
AVG(net_out) AS net_out_avg
FROM (
  SELECT machine_name,
    COALESCE(cpu_user, 0.0) AS cpu,
    COALESCE(bytes_in, 0.0) AS net_in,
    COALESCE(bytes_out, 0.0) AS net_out
  FROM logs1
  WHERE machine_name IN ('anansi','aragog','urd')
    AND log_time >= TIMESTAMP '2017-01-11 00:00:00'
) AS r
GROUP BY machine_name;

```

-- Q1.2: Which computer lab machines have been offline in the past day?

```

SELECT machine_name,
  log_time
FROM logs1
WHERE (machine_name LIKE 'cslab%' OR
      machine_name LIKE 'mslab%')
  AND load_one IS NULL
  AND log_time >= TIMESTAMP '2017-01-10 00:00:00'
ORDER BY machine_name,
  log_time;

```

-- Q1.3: What are the hourly average metrics during the past 10 days for a specific workstation?

```

SELECT dt,
  hr,
  AVG(load_fifteen) AS load_fifteen_avg,
  AVG(load_five) AS load_five_avg,
  AVG(load_one) AS load_one_avg,
  AVG(mem_free) AS mem_free_avg,
  AVG(swap_free) AS swap_free_avg
FROM (
  SELECT CAST(log_time AS DATE) AS dt,
    EXTRACT(HOUR FROM log_time) AS hr,
    load_fifteen,
    load_five,
    load_one,
    mem_free,
    swap_free
  FROM logs1
  WHERE machine_name = 'babbage'
    AND load_fifteen IS NOT NULL
    AND load_five IS NOT NULL
    AND load_one IS NOT NULL
    AND mem_free IS NOT NULL
    AND swap_free IS NOT NULL
    AND log_time >= TIMESTAMP '2017-01-01 00:00:00'
) AS r
GROUP BY dt,
  hr
ORDER BY dt,
  hr;

```

-- Q1.4: Over 1 month, how often was each server blocked on disk I/O?

```

SELECT machine_name,
  COUNT(*) AS spikes
FROM logs1
WHERE machine_group = 'Servers'
  AND cpu_wio > 0.99
  AND log_time >= TIMESTAMP '2016-12-01 00:00:00'
  AND log_time < TIMESTAMP '2017-01-01 00:00:00'
GROUP BY machine_name
ORDER BY spikes DESC
LIMIT 10;

```

-- Q1.5: Which externally reachable VMs have run low on memory?

```

SELECT machine_name,
  ...

```

```

at,
MIN(mem_free) AS mem_free_min
FROM (
  SELECT machine_name,
    CAST(log_time AS DATE) AS dt,
    mem_free
  FROM logs1
  WHERE machine_group = 'DMZ'
    AND mem_free IS NOT NULL
) AS r
GROUP BY machine_name,
  dt
HAVING MIN(mem_free) < 10000
ORDER BY machine_name,
  dt;

```

-- Q1.6: What is the total hourly network traffic across all file servers?

```

SELECT dt,
  hr,
  SUM(net_in) AS net_in_sum,
  SUM(net_out) AS net_out_sum,
  SUM(net_in) + SUM(net_out) AS both_sum
FROM (
  SELECT CAST(log_time AS DATE) AS dt,
    EXTRACT(HOUR FROM log_time) AS hr,
    COALESCE(bytes_in, 0.0) / 1000000000.0 AS net_in,
    COALESCE(bytes_out, 0.0) / 1000000000.0 AS net_out
  FROM logs1
  WHERE machine_name IN ('allsorts','andes','bigred','blackjack','bonbon',
    'cadbury','chiclets','cotton','crows','dove','fireball','hearts','huey',
    'lindt','milkduds','milkyway','mnmm','necco','nerds','orbit','peeps',
    'poprocks','razzles','runts','smarties','smuggler','spree','stride',
    'tootsie','trident','wrigley','york')
) AS r
GROUP BY dt,
  hr
ORDER BY both_sum DESC
LIMIT 10;

```

-- Q2.1: Which requests have caused server errors within the past 2 weeks?

```

SELECT *
FROM logs2
WHERE status_code >= 500
  AND log_time >= TIMESTAMP '2012-12-18 00:00:00'
ORDER BY log_time;

```

-- Q2.2: During a specific 2-week period, was the user password file leaked?

```

SELECT *
FROM logs2
WHERE status_code >= 200
  AND status_code < 300
  AND request LIKE '%/etc/passwd%'
  AND log_time >= TIMESTAMP '2012-05-06 00:00:00'
  AND log_time < TIMESTAMP '2012-05-20 00:00:00';

```

-- Q2.3: What was the average path depth for top-level requests in the past month?

```

SELECT top_level,
  AVG(LENGTH(request) - LENGTH(REPLACE(request, '/', ''))) AS depth_avg
FROM (
  SELECT SUBSTRING(request FROM 1 FOR len) AS top_level,
    request
  FROM (
    SELECT POSITION(SUBSTRING(request FROM 2), '/') AS len,
      request
    FROM logs2
    WHERE status_code >= 200
      AND status_code < 300
      AND log_time >= TIMESTAMP '2012-12-01 00:00:00'
  ) AS r
)
```

```
    WHERE len > 0
) AS s
WHERE top_level IN ('/about','/courses','/degrees','/events',
                     '/grad','/industry','/news','/people',
                     '/publications','/research','/teaching','/ugrad')
GROUP BY top_level
ORDER BY top_level;
```

-- Q2.4: During the last 3 months, which clients have made an excessive number of requests?

```
SELECT client_ip,
       COUNT(*) AS num_requests
  FROM logs2
 WHERE log_time >= TIMESTAMP '2012-10-01 00:00:00'
 GROUP BY client_ip
 HAVING COUNT(*) >= 100000
 ORDER BY num_requests DESC;
```

-- Q2.5: What are the daily unique visitors?

```
SELECT dt,
       COUNT(DISTINCT client_ip)
  FROM (
    SELECT CAST(log_time AS DATE) AS dt,
           client_ip
      FROM logs2
  ) AS r
 GROUP BY dt
 ORDER BY dt;
```

-- Q2.6: What are the average and maximum data transfer rates (Gbps)?

```
SELECT AVG(transfer) / 125000000.0 AS transfer_avg,
       MAX(transfer) / 125000000.0 AS transfer_max
  FROM (
    SELECT log_time,
           SUM(object_size) AS transfer
      FROM logs2
     GROUP BY log_time
  ) AS r;
```

-- Q3.1: Did the indoor temperature reach freezing over the weekend?

```
SELECT *
  FROM logs3
 WHERE event_type = 'temperature'
   AND event_value <= 32.0
   AND log_time >= '2019-11-29 17:00:00.000';
```

-- Q3.4: Over the past 6 months, how frequently were each door opened?

```
SELECT device_name,
       device_floor,
       COUNT(*) AS ct
  FROM logs3
 WHERE event_type = 'door_open'
   AND log_time >= '2019-06-01 00:00:00.000'
 GROUP BY device_name,
          device_floor
 ORDER BY ct DESC;
```

-- Q3.5: Where in the building do large temperature variations occur in winter and summer?

```
WITH temperature AS (
  SELECT dt,
         device_name,
         device_type,
         device_floor
    FROM (
      SELECT dt,
             hr,
```

```

device_name,
device_type,
device_floor,
AVG(event_value) AS temperature_hourly_avg
FROM (
    SELECT CAST(log_time AS DATE) AS dt,
        EXTRACT(HOUR FROM log_time) AS hr,
        device_name,
        device_type,
        device_floor,
        event_value
    FROM logs3
    WHERE event_type = 'temperature'
) AS r
GROUP BY dt,
    hr,
    device_name,
    device_type,
    device_floor
) AS s
GROUP BY dt,
    device_name,
    device_type,
    device_floor
HAVING MAX(temperature_hourly_avg) - MIN(temperature_hourly_avg) >= 25.0
)
SELECT DISTINCT device_name,
    device_type,
    device_floor,
    'WINTER'
FROM temperature
WHERE dt >= DATE '2018-12-01'
    AND dt < DATE '2019-03-01'
UNION
SELECT DISTINCT device_name,
    device_type,
    device_floor,
    'SUMMER'
FROM temperature
WHERE dt >= DATE '2019-06-01'
    AND dt < DATE '2019-09-01';

```

-- Q3.6: For each device category, what are the monthly power consumption metrics?

```

SELECT yr,
    mo,
    SUM(coffee_hourly_avg) AS coffee_monthly_sum,
    AVG(coffee_hourly_avg) AS coffee_monthly_avg,
    SUM(printer_hourly_avg) AS printer_monthly_sum,
    AVG(printer_hourly_avg) AS printer_monthly_avg,
    SUM(projector_hourly_avg) AS projector_monthly_sum,
    AVG(projector_hourly_avg) AS projector_monthly_avg,
    SUM(vending_hourly_avg) AS vending_monthly_sum,
    AVG(vending_hourly_avg) AS vending_monthly_avg
FROM (
    SELECT dt,
        yr,
        mo,
        hr,
        AVG(coffee) AS coffee_hourly_avg,
        AVG(printer) AS printer_hourly_avg,
        AVG(projector) AS projector_hourly_avg,
        AVG(vending) AS vending_hourly_avg
    FROM (
        SELECT CAST(log_time AS DATE) AS dt,
            EXTRACT(YEAR FROM log_time) AS yr,
            EXTRACT(MONTH FROM log_time) AS mo,
            EXTRACT(HOUR FROM log_time) AS hr,
            CASE WHEN device_name LIKE 'coffee%' THEN event_value END AS coffee,
            CASE WHEN device_name LIKE 'printer%' THEN event_value END AS printer,
            CASE WHEN device_name LIKE 'projector%' THEN event_value END AS projector,
            CASE WHEN device_name LIKE 'vending%' THEN event_value END AS vending
        FROM logs3
        WHERE device_type = 'meter'
    ) AS r
    GROUP BY dt,

```

```
    yr,  
    mo,  
    hr  
) AS s  
GROUP BY yr,  
        mo  
ORDER BY yr,  
        mo;
```

The data is also available for interactive queries in the [Playground](#), [example](#).

## New York Taxi Data

This dataset can be obtained in two ways:

- import from raw data
- download of prepared partitions

### How to Import the Raw Data

See <https://github.com/toddwschneider/nyc-taxi-data> and <http://tech.marksblogg.com/billion-nyc-taxi-rides-redshift.html> for the description of a dataset and instructions for downloading.

Downloading will result in about 227 GB of uncompressed data in CSV files. The download takes about an hour over a 1 Gbit connection (parallel downloading from s3.amazonaws.com recovers at least half of a 1 Gbit channel).

Some of the files might not download fully. Check the file sizes and re-download any that seem doubtful.

Some of the files might contain invalid rows. You can fix them as follows:

```
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
```

Then the data must be pre-processed in PostgreSQL. This will create selections of points in the polygons (to match points on the map with the boroughs of New York City) and combine all the data into a single denormalized flat table by using a JOIN. To do this, you will need to install PostgreSQL with PostGIS support.

Be careful when running `initialize_database.sh` and manually re-check that all the tables were created correctly.

It takes about 20-30 minutes to process each month's worth of data in PostgreSQL, for a total of about 48 hours.

You can check the number of downloaded rows as follows:

```
$ time psql nyc-taxi-data -c "SELECT count(*) FROM trips;"  
### Count  
1298979494  
(1 row)  
  
real 7m9.164s
```

(This is slightly more than 1.1 billion rows reported by Mark Litwintschik in a series of blog posts.)

The data in PostgreSQL uses 370 GB of space.

Exporting the data from PostgreSQL:

```
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';
```

The data snapshot is created at a speed of about 50 MB per second. While creating the snapshot, PostgreSQL reads from the disk at a speed of about 28 MB per second. This takes about 5 hours. The resulting TSV file is 590612904969 bytes.

Create a temporary table in ClickHouse:

```
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_boroc2010 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_boroc2010 Nullable(String),
    dropoff_cdeligibil Nullable(String),
    dropoff_ntacode  Nullable(String),
    dropoff_ntaname  Nullable(String),
    dropoff_puma     Nullable(String)
) ENGINE = Log;
```

It is needed for converting fields to more correct data types and, if possible, to eliminate NULLs.

```
$ time clickhouse-client --query="INSERT INTO trips FORMAT TabSeparated" < trips.tsv
real 75m56.214s
```

Data is read at a speed of 112-140 Mb/second.

Loading data into a Log type table in one stream took 76 minutes.

The data in this table uses 142 GB.

(Importing data directly from Postgres is also possible using COPY ... TO PROGRAM.)

Unfortunately, all the fields associated with the weather (precipitation...average\_wind\_speed) were filled with NULL. Because of this, we will remove them from the final data set.

To start, we'll create a table on a single server. Later we will make the table distributed.

Create and populate a summary table:

```
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_boroc2010, '0000000'), 7) AS pickup_boroc2010,
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-Tottenvile' = 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-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, 'Lauriston' = 95, 'Lefferts
```

Parkway' = 91, New Gardens' = 92, New Gardens Hills' = 93, Kingsbridge Heights' = 94, Laurelhurst' = 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\_boroname,  
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-Hugenot-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-Carrington Beach-Manhattan

```

'Throgs Neck-Eugewater Park' = 140, 'Seagate-Coney Island' = 149, 'Sleepy Hollow 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

This takes 3030 seconds at a speed of about 428,000 rows per second.

To load it faster, you can create the table with the `Log` engine instead of `MergeTree`. In this case, the download works faster than 200 seconds.

The table uses 126 GB of disk space.

```
SELECT formatReadableSize(sum(bytes)) FROM system.parts WHERE table = 'trips_mergetree' AND active
```

```
formatReadableSize(sum(bytes))—  
126.18 GiB |
```

Among other things, you can run the `OPTIMIZE` query on `MergeTree`. But it's not required since everything will be fine without it.

## Download of Prepared Partitions

```
$ curl -O https://datasets.clickhouse.com/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"
```

### Info

If you will run the queries described below, you have to use the full table name, `datasets.trips_mergetree`.

## Results on Single Server

Q1:

```
SELECT cab_type, count(*) FROM trips_mergetree GROUP BY cab_type
```

0.490 seconds.

Q2:

```
SELECT passenger_count, avg(total_amount) FROM trips_mergetree GROUP BY passenger_count
```

1.224 seconds.

Q3:

```
SELECT passenger_count, toYear(pickup_date) AS year, count(*) FROM trips_mergetree GROUP BY passenger_count, year
```

2.104 seconds.

Q4:

```
SELECT passenger_count, toYear(pickup_date) AS year, round(trip_distance) AS distance, count(*)  
FROM trips_mergetree  
GROUP BY passenger_count, year, distance  
ORDER BY year, count(*) DESC
```

3.593 seconds.

The following server was used:

Two Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz, 16 physical kernels total, 128 GiB RAM, 8x6 TB HD on hardware RAID-5

Execution time is the best of three runs. But starting from the second run, queries read data from the file system cache. No further caching occurs: the data is read out and processed in each run.

Creating a table on three servers:

On each server:

```
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_boroc2010 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, '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), 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)
```

On the source server:

```
CREATE TABLE trips_mergetree_x3 AS trips_mergetree_third ENGINE = Distributed(perftest, default, trips_mergetree_third, rand())
```

The following query redistributes data:

```
INSERT INTO trips_mergetree_x3 SELECT * FROM trips_mergetree
```

This takes 2454 seconds.

On three servers:

Q1: 0.212 seconds.  
Q2: 0.438 seconds.  
Q3: 0.733 seconds.  
Q4: 1.241 seconds.

No surprises here, since the queries are scaled linearly.

We also have the results from a cluster of 140 servers:

Q1: 0.028 sec.  
Q2: 0.043 sec.  
Q3: 0.051 sec.  
Q4: 0.072 sec.

In this case, the query processing time is determined above all by network latency.

We ran queries using a client located in a Yandex datacenter in Finland on a cluster in Russia, which added about 20 ms of latency.

## Summary

servers	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

## Crowdsourced air traffic data from The OpenSky Network 2020

"The data in this dataset is derived and cleaned from the full OpenSky dataset to illustrate the development of air traffic during the COVID-19 pandemic. It spans all flights seen by the network's more than 2500 members since 1 January 2019. More data will be periodically included in the dataset until the end of the COVID-19 pandemic".

Source: <https://zenodo.org/record/5092942#.YRBCyTpRXYd>

Martin Strohmeier, Xavier Olive, Jannis Lübbe, Matthias Schäfer, and Vincent Lenders  
"Crowdsourced air traffic data from the OpenSky Network 2019–2020"  
Earth System Science Data 13(2), 2021  
<https://doi.org/10.5194/essd-13-357-2021>

## Download the Dataset

Run the command:

```
wget -O- https://zenodo.org/record/5092942 | grep -oP  
'https://zenodo.org/record/5092942/files/flightlist_\d+_d+\.csv\.gz' | xargs wget
```

Download will take about 2 minutes with good internet connection. There are 30 files with total size of 4.3 GB.

## Create the Table

```
CREATE TABLE opensky  
(  
    callsign String,  
    number String,  
    icao24 String,  
    registration String,  
    typecode String,  
    origin String,  
    destination String,  
    firstseen DateTime,  
    lastseen DateTime,  
    day DateTime,  
    latitude_1 Float64,  
    longitude_1 Float64,  
    altitude_1 Float64,  
    latitude_2 Float64,  
    longitude_2 Float64,  
    altitude_2 Float64  
) ENGINE = MergeTree ORDER BY (origin, destination, callsign);
```

## Import Data

Upload data into ClickHouse in parallel:

```
ls -1 flightlist_*.csv.gz | xargs -P100 -I{} bash -c 'gzip -c -d "{}" | clickhouse-client --date_time_input_format  
best_effort --query "INSERT INTO opensky FORMAT CSVWithNames"'
```

- Here we pass the list of files (`ls -1 flightlist_*.csv.gz`) to `xargs` for parallel processing.  
`xargs -P100` specifies to use up to 100 parallel workers but as we only have 30 files, the number of workers will be only 30.
- For every file, `xargs` will run a script with `bash -c`. The script has substitution in form of `{}` and the `xargs` command will substitute the filename to it (we have asked it for `xargs` with `-I{}`).
- The script will decompress the file (`gzip -c -d "{}"`) to standard output (`-c` parameter) and the output is redirected to `clickhouse-client`.
- We also asked to parse `DateTime` fields with extended parser (`--date_time_input_format best_effort`) to recognize ISO-8601 format with timezone offsets.

Finally, `clickhouse-client` will do insertion. It will read input data in `CSVWithNames` format.

Parallel upload takes 24 seconds.

If you don't like parallel upload, here is sequential variant:

```
for file in flightlist_*.csv.gz; do gzip -c -d "$file" | clickhouse-client --date_time_input_format best_effort --query "INSERT INTO opensky FORMAT CSVWithNames"; done
```

## Validate the Data

Query:

```
SELECT count() FROM opensky;
```

Result:

```
count()
66010819
```

The size of dataset in ClickHouse is just 2.66 GiB, check it.

Query:

```
SELECT formatReadableSize(total_bytes) FROM system.tables WHERE name = 'opensky';
```

Result:

```
formatReadableSize(total_bytes)
2.66 GiB
```

## Run Some Queries

Total distance travelled is 68 billion kilometers.

Query:

```
SELECT formatReadableQuantity(sum(geoDistance(longitude_1, latitude_1, longitude_2, latitude_2)) / 1000) FROM opensky;
```

Result:

```
formatReadableQuantity(divide(sum(geoDistance(longitude_1, latitude_1, longitude_2, latitude_2)), 1000))
68.72 billion
```

Average flight distance is around 1000 km.

Query:

```
SELECT avg(geoDistance(longitude_1, latitude_1, longitude_2, latitude_2)) FROM opensky;
```

Result:

```

avg(geoDistance(longitude_1, latitude_1, longitude_2, latitude_2)) ->
1041090.6465708319 |

```

## Most busy origin airports and the average distance seen

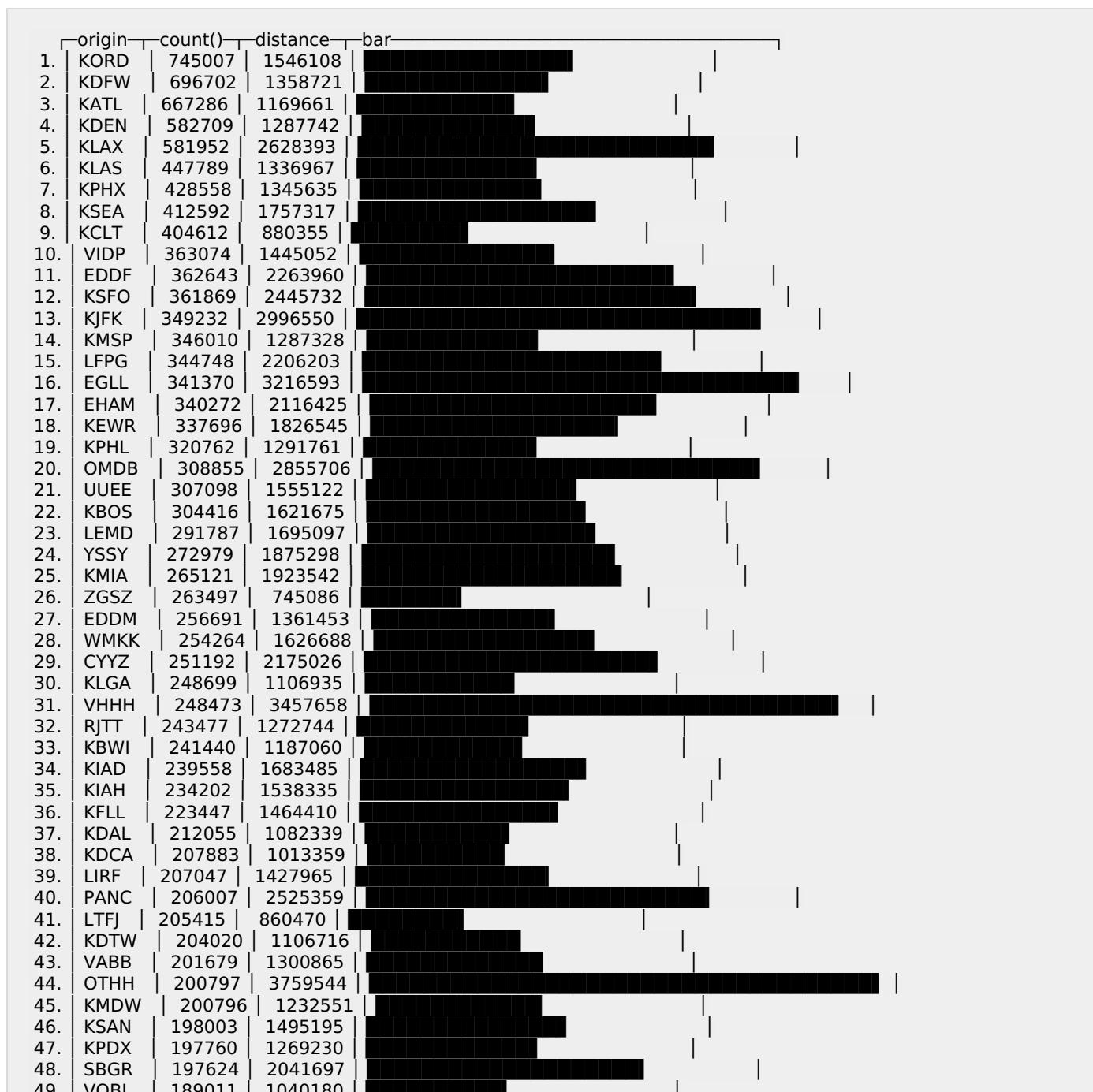
Query:

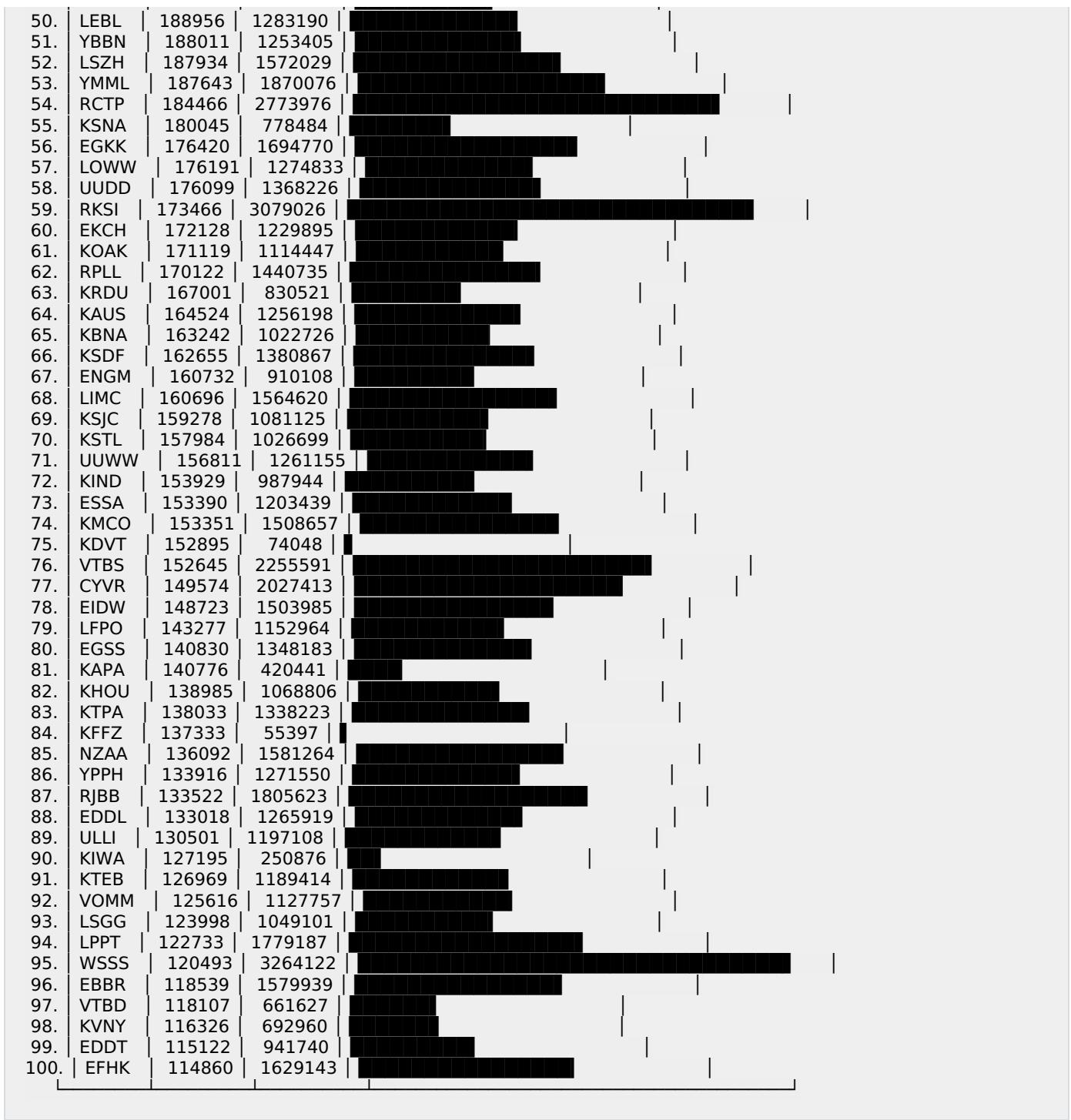
```

SELECT
    origin,
    count(),
    round(avg(geoDistance(longitude_1, latitude_1, longitude_2, latitude_2))) AS distance,
    bar(distance, 0, 10000000, 100) AS bar
FROM opensky
WHERE origin != ''
GROUP BY origin
ORDER BY count() DESC
LIMIT 100;

```

Result:





## Number of flights from three major Moscow airports, weekly

Query:

```

SELECT
    toMonday(day) AS k,
    count() AS c,
    bar(c, 0, 10000, 100) AS bar
FROM opensky
WHERE origin IN ('UUUE', 'UUDD', 'UUWW')
GROUP BY k
ORDER BY k ASC;

```

Result:



2.	2019-01-07	6302
3.	2019-01-14	5701
4.	2019-01-21	5638
5.	2019-01-28	5731
6.	2019-02-04	5683
7.	2019-02-11	5759
8.	2019-02-18	5736
9.	2019-02-25	5873
10.	2019-03-04	5965
11.	2019-03-11	5900
12.	2019-03-18	5823
13.	2019-03-25	5899
14.	2019-04-01	6043
15.	2019-04-08	6098
16.	2019-04-15	6196
17.	2019-04-22	6486
18.	2019-04-29	6682
19.	2019-05-06	6739
20.	2019-05-13	6600
21.	2019-05-20	6575
22.	2019-05-27	6786
23.	2019-06-03	6872
24.	2019-06-10	7045
25.	2019-06-17	7045
26.	2019-06-24	6852
27.	2019-07-01	7248
28.	2019-07-08	7284
29.	2019-07-15	7142
30.	2019-07-22	7108
31.	2019-07-29	7251
32.	2019-08-05	7403
33.	2019-08-12	7457
34.	2019-08-19	7502
35.	2019-08-26	7540
36.	2019-09-02	7237
37.	2019-09-09	7328
38.	2019-09-16	5566
39.	2019-09-23	7049



88.	2020-08-31	4667	
89.	2020-09-07	4643	
90.	2020-09-14	4594	
91.	2020-09-21	4478	
92.	2020-09-28	4382	
93.	2020-10-05	4261	
94.	2020-10-12	4243	
95.	2020-10-19	3941	
96.	2020-10-26	3616	
97.	2020-11-02	3586	
98.	2020-11-09	3403	
99.	2020-11-16	3336	
100.	2020-11-23	3230	
101.	2020-11-30	3183	
102.	2020-12-07	3285	
103.	2020-12-14	3367	
104.	2020-12-21	3748	
105.	2020-12-28	3986	
106.	2021-01-04	3906	
107.	2021-01-11	3425	
108.	2021-01-18	3144	
109.	2021-01-25	3115	
110.	2021-02-01	3285	
111.	2021-02-08	3321	
112.	2021-02-15	3475	
113.	2021-02-22	3549	
114.	2021-03-01	3755	
115.	2021-03-08	3080	
116.	2021-03-15	3789	
117.	2021-03-22	3804	
118.	2021-03-29	4238	
119.	2021-04-05	4307	
120.	2021-04-12	4225	
121.	2021-04-19	4391	
122.	2021-04-26	4868	
123.	2021-05-03	4977	
124.	2021-05-10	5164	
125.	2021-05-17	4986	
126.	2021-05-24	5024	
127.	2021-05-31	4824	
128.	2021-06-07	5652	
129.	2021-06-14	5613	
130.	2021-06-21	6061	
131.	2021-06-28	2554	

## Online Playground

You can test other queries to this data set using the interactive resource [Online Playground](#). For example, [like this](#). However, please note that you cannot create temporary tables here.

## UK Property Price Paid

The dataset contains data about prices paid for real-estate property in England and Wales. The data is available since year 1995.

The size of the dataset in uncompressed form is about 4 GiB and it will take about 278 MiB in ClickHouse.

Source: <https://www.gov.uk/government/statistical-data-sets/price-paid-data-downloads>

Description of the fields: <https://www.gov.uk/guidance/about-the-price-paid-data>

Contains HM Land Registry data © Crown copyright and database right 2021. This data is licensed under the Open Government Licence v3.0.

## Download the Dataset

Run the command:

```
wget http://prod.publicdata.landregistry.gov.uk.s3-website-eu-west-1.amazonaws.com/pp-complete.csv
```

Download will take about 2 minutes with good internet connection.

## Create the Table

```
CREATE TABLE uk_price_paid
(
    price UInt32,
    date Date,
    postcode1 LowCardinality(String),
    postcode2 LowCardinality(String),
    type Enum8('terraced' = 1, 'semi-detached' = 2, 'detached' = 3, 'flat' = 4, 'other' = 0),
    is_new UInt8,
    duration Enum8('freehold' = 1, 'leasehold' = 2, 'unknown' = 0),
    addr1 String,
    addr2 String,
    street LowCardinality(String),
    locality LowCardinality(String),
    town LowCardinality(String),
    district LowCardinality(String),
    county LowCardinality(String),
    category UInt8
) ENGINE = MergeTree ORDER BY (postcode1, postcode2, addr1, addr2);
```

## Preprocess and Import Data

We will use `clickhouse-local` tool for data preprocessing and `clickhouse-client` to upload it.

In this example, we define the structure of source data from the CSV file and specify a query to preprocess the data with `clickhouse-local`.

The preprocessing is:

- splitting the postcode to two different columns `postcode1` and `postcode2` that is better for storage and queries;
- converting the time field to date as it only contains 00:00 time;
- ignoring the `UUId` field because we don't need it for analysis;
- transforming `type` and `duration` to more readable Enum fields with function `transform`;
- transforming `is_new` and `category` fields from single-character string (Y/N and A/B) to `UInt8` field with 0 and 1.

Preprocessed data is piped directly to `clickhouse-client` to be inserted into ClickHouse table in streaming fashion.

```

clickhouse-local --input-format CSV --structure '
    uuid String,
    price UInt32,
    time DateTime,
    postcode String,
    a String,
    b String,
    c String,
    addr1 String,
    addr2 String,
    street String,
    locality String,
    town String,
    district String,
    county String,
    d String,
    e String
' --query "
WITH splitByChar(' ', postcode) AS p
SELECT
    price,
    toDate(time) AS date,
    p[1] AS postcode1,
    p[2] AS postcode2,
    transform(a, ['T', 'S', 'D', 'F', 'O'], ['terraced', 'semi-detached', 'detached', 'flat', 'other']) AS type,
    b = 'Y' AS is_new,
    transform(c, ['F', 'L', 'U'], ['freehold', 'leasehold', 'unknown']) AS duration,
    addr1,
    addr2,
    street,
    locality,
    town,
    district,
    county,
    d = 'B' AS category
FROM table" --date_time input_format best_effort < pp-complete.csv | clickhouse-client --query "INSERT INTO
uk_price_paid FORMAT TSV"

```

It will take about 40 seconds.

## Validate the Data

Query:

```
SELECT count() FROM uk_price_paid;
```

Result:

count()
26321785

The size of dataset in ClickHouse is just 278 MiB, check it.

Query:

```
SELECT formatReadableSize(total_bytes) FROM system.tables WHERE name = 'uk_price_paid';
```

Result:

formatReadableSize(total_bytes)
278.80 MiB

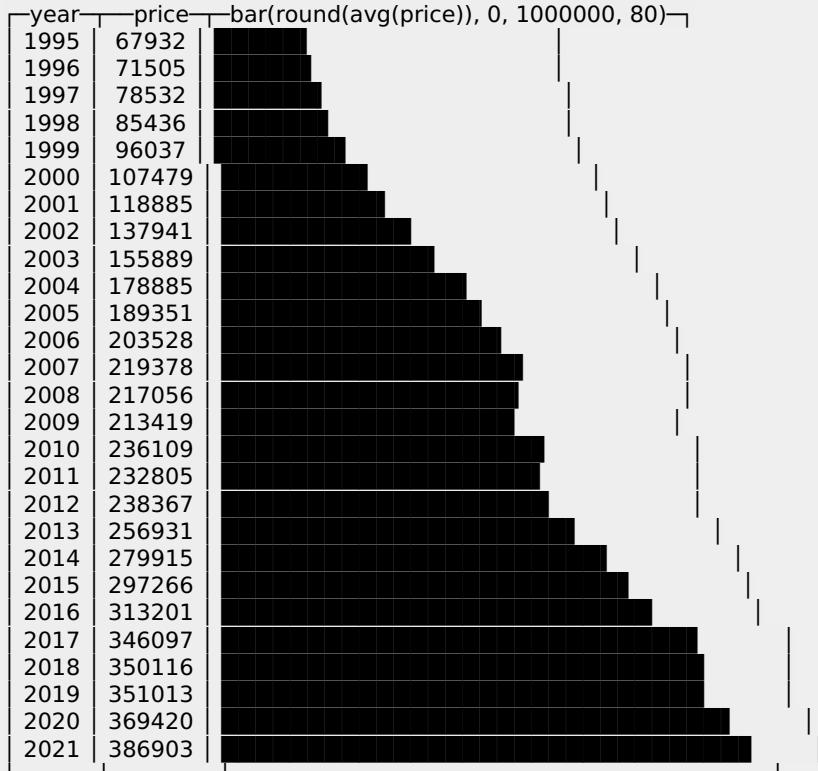
# Run Some Queries

## Query 1. Average Price Per Year

Query:

```
SELECT toYear(date) AS year, round(avg(price)) AS price, bar(price, 0, 1000000, 80) FROM uk_price_paid GROUP BY year ORDER BY year;
```

Result:

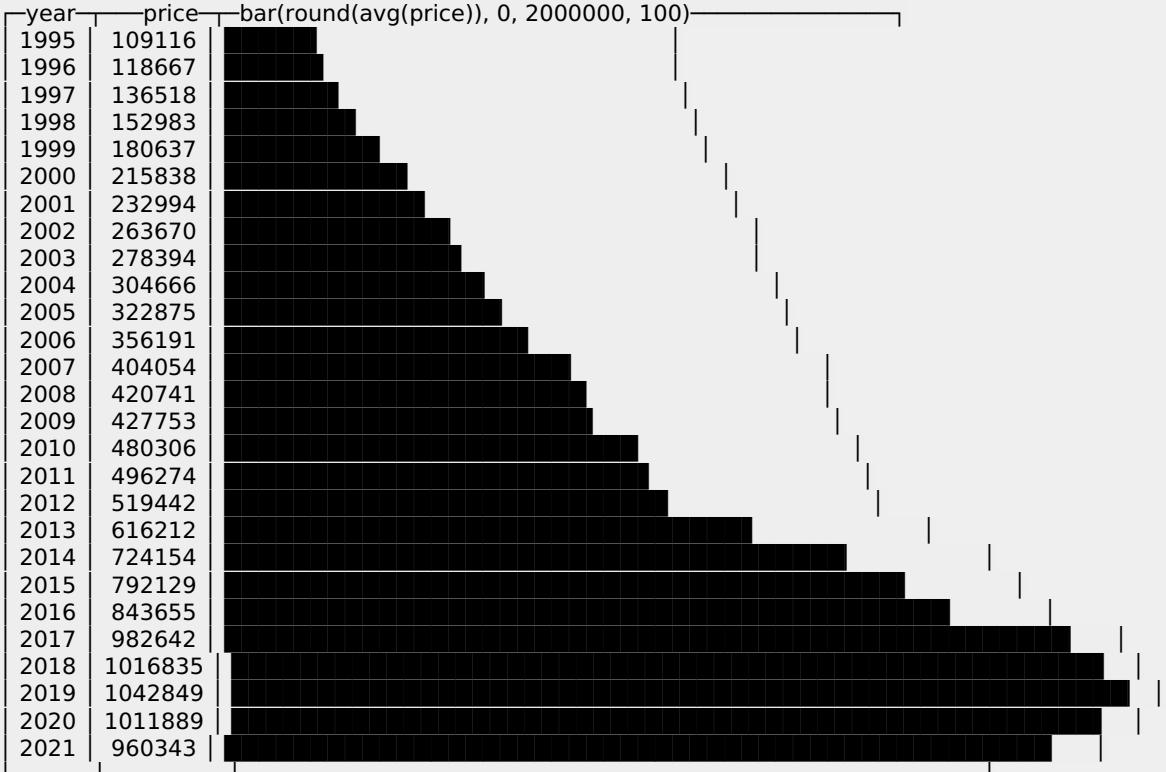


## Query 2. Average Price per Year in London

Query:

```
SELECT toYear(date) AS year, round(avg(price)) AS price, bar(price, 0, 2000000, 100) FROM uk_price_paid WHERE town = 'LONDON' GROUP BY year ORDER BY year;
```

Result:



Something happened in 2013. I don't have a clue. Maybe you have a clue what happened in 2020?

### Query 3. The Most Expensive Neighborhoods

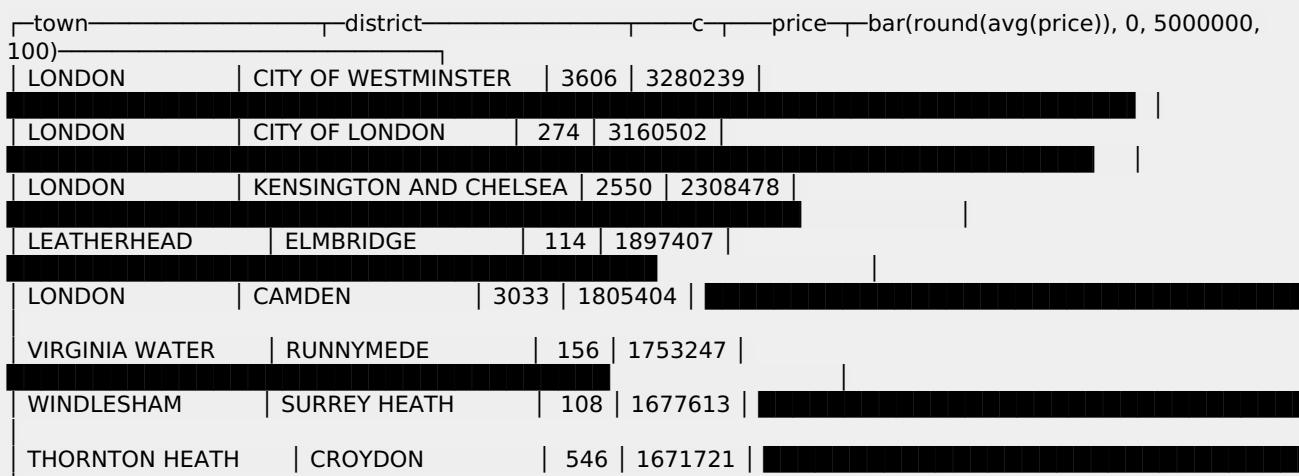
Query:

```

SELECT
    town,
    district,
    count() AS c,
    round(avg(price)) AS price,
    bar(price, 0, 5000000, 100)
FROM uk_price_paid
WHERE date >= '2020-01-01'
GROUP BY
    town,
    district
HAVING c >= 100
ORDER BY price DESC
LIMIT 100;

```

Result:



BARNET	ENFIELD	124   1505840   [REDACTED]
COBHAM	ELMBRIDGE	387   1237250   [REDACTED]
LONDON	ISLINGTON	2668   1236980   [REDACTED]
OXFORD	SOUTH OXFORDSHIRE	321   1220907   [REDACTED]
LONDON	RICHMOND UPON THAMES	704   1215551   [REDACTED]
LONDON	HOUNSLOW	671   1207493   [REDACTED]
ASCOT	WINDSOR AND MAIDENHEAD	407   1183299   [REDACTED]
BEACONSFIELD	BUCKINGHAMSHIRE	330   1175615   [REDACTED]
RICHMOND	RICHMOND UPON THAMES	874   1110444   [REDACTED]
LONDON	HAMMERSMITH AND FULHAM	3086   1053983   [REDACTED]
SURBITON	ELMBRIDGE	100   1011800   [REDACTED]
RADLETT	HERTSMERE	283   1011712   [REDACTED]
SALCOMBE	SOUTH HAMS	127   1011624   [REDACTED]
WEYBRIDGE	ELMBRIDGE	655   1007265   [REDACTED]
ESHER	ELMBRIDGE	485   986581   [REDACTED]
LEATHERHEAD	GUILDFORD	202   977320   [REDACTED]
BURFORD	WEST OXFORDSHIRE	111   966893   [REDACTED]
BROCKENHURST	NEW FOREST	129   956675   [REDACTED]
HINDHEAD	WAVERLEY	137   953753   [REDACTED]
GERRARDS CROSS	BUCKINGHAMSHIRE	419   951121   [REDACTED]
EAST MOLESEY	ELMBRIDGE	192   936769   [REDACTED]
CHALFONT ST GILES	BUCKINGHAMSHIRE	146   925515   [REDACTED]
LONDON	TOWER HAMLETS	4388   918304   [REDACTED]
OLNEY	MILTON KEYNES	235   910646   [REDACTED]
HENLEY-ON-THAMES	SOUTH OXFORDSHIRE	540   902418   [REDACTED]
LONDON	SOUTHWARK	3885   892997   [REDACTED]
KINGSTON UPON THAMES	KINGSTON UPON THAMES	960   885969   [REDACTED]
LONDON	EALING	2658   871755   [REDACTED]
CRANBROOK	TUNBRIDGE WELLS	431   862348   [REDACTED]
LONDON	MERTON	2099   859118   [REDACTED]
BELVEDERE	BEXLEY	346   842423   [REDACTED]
GUILDFORD	WAVERLEY	143   841277   [REDACTED]
HARPENDEN	ST ALBANS	657   841216   [REDACTED]
LONDON	HACKNEY	3307   837090   [REDACTED]
LONDON	WANDSWORTH	6566   832663   [REDACTED]
MAIDENHEAD	BUCKINGHAMSHIRE	123   824299   [REDACTED]
KINGS LANGLEY	DACORUM	145   821331   [REDACTED]
BERKHAMSTED	DACORUM	543   818415   [REDACTED]
GREAT MISSENDEN	BUCKINGHAMSHIRE	226   802807   [REDACTED]
BILLINGSHURST	CHICHESTER	144   797829   [REDACTED]
WOKING	GUILDFORD	176   793494   [REDACTED]
STOCKBRIDGE	TEST VALLEY	178   793269   [REDACTED]
EPSOM	REIGATE AND BANSTEAD	172   791862   [REDACTED]

TONBRIDGE	TUNBRIDGE WELLS	360	787876	[REDACTED]
TEDDINGTON	RICHMOND UPON THAMES	595	786492	[REDACTED]
TWICKENHAM	RICHMOND UPON THAMES	1155	786193	[REDACTED]
LYNDHURST	NEW FOREST	102	785593	[REDACTED]
LONDON	LAMBETH	5228	774574	[REDACTED]
LONDON	BARNET	3955	773259	[REDACTED]
OXFORD	VALE OF WHITE HORSE	353	772088	[REDACTED]
TONBRIDGE	MAIDSTONE	305	770740	[REDACTED]
LUTTERWORTH	HARBOROUGH	538	768634	[REDACTED]
WOODSTOCK	WEST OXFORDSHIRE	140	766037	[REDACTED]
MIDHURST	CHICHESTER	257	764815	[REDACTED]
MARLOW	BUCKINGHAMSHIRE	327	761876	[REDACTED]
LONDON	NEWHAM	3237	761784	[REDACTED]
ALDERLEY EDGE	CHESHIRE EAST	178	757318	[REDACTED]
LUTON	CENTRAL BEDFORDSHIRE	212	754283	[REDACTED]
PETWORTH	CHICHESTER	154	754220	[REDACTED]
ALRESFORD	WINCHESTER	219	752718	[REDACTED]
POTTERS BAR	WELWYN HATFIELD	174	748465	[REDACTED]
HASLEMERE	CHICHESTER	128	746907	[REDACTED]
TADWORTH	REIGATE AND BANSTEAD	502	743252	[REDACTED]
THAMES DITTON	ELMBRIDGE	244	741913	[REDACTED]
REIGATE	REIGATE AND BANSTEAD	581	738198	[REDACTED]
BOURNE END	BUCKINGHAMSHIRE	138	735190	[REDACTED]
SEVENOAKS	SEVENOAKS	1156	730018	[REDACTED]
OXTED	TANDRIDGE	336	729123	[REDACTED]
INGATESTONE	BRENTWOOD	166	728103	[REDACTED]
LONDON	BRENT	2079	720605	[REDACTED]
LONDON	HARINGEY	3216	717780	[REDACTED]
PURLEY	CROYDON	575	716108	[REDACTED]
WELWYN	WELWYN HATFIELD	222	710603	[REDACTED]
RICKMANSWORTH	THREE RIVERS	798	704571	[REDACTED]
BANSTEAD	REIGATE AND BANSTEAD	401	701293	[REDACTED]
CHIGWELL	EPPING FOREST	261	701203	[REDACTED]
PINNER	HARROW	528	698885	[REDACTED]
HASLEMERE	WAVERLEY	280	696659	[REDACTED]
SLOUGH	BUCKINGHAMSHIRE	396	694917	[REDACTED]
WALTON-ON-THAMES	ELMBRIDGE	946	692395	[REDACTED]
READING	SOUTH OXFORDSHIRE	318	691988	[REDACTED]
NORTHWOOD	HILLINGDON	271	690643	[REDACTED]
FELTHAM	HOUNSLAW	763	688595	[REDACTED]
ASHTEAD	MOLE VALLEY	303	687923	[REDACTED]
BARNET	BARNET	975	686980	[REDACTED]
WOKING	SURREY HEATH	283	686669	[REDACTED]
MALMESBURY	WILTSHIRE	323	683324	[REDACTED]
AMERSHAM	BUCKINGHAMSHIRE	496	680962	[REDACTED]
CHISLEHURST	BROMLEY	430	680209	[REDACTED]
HYTHE	FOLKESTONE AND HYTHE	490	676908	[REDACTED]
MAYFIELD	WEALDEN	101	676210	[REDACTED]
ASCOT	BRACKNELL FOREST	168	676004	[REDACTED]



# Let's Speed Up Queries Using Projections

Projections allow to improve queries speed by storing pre-aggregated data.

## Build a Projection

Create an aggregate projection by dimensions `toYear(date)`, `district`, `town`:

```
ALTER TABLE uk_price_paid
  ADD PROJECTION projection_by_year_district_town
(
  SELECT
    toYear(date),
    district,
    town,
    avg(price),
    sum(price),
    count()
  GROUP BY
    toYear(date),
    district,
    town
);
```

Populate the projection for existing data (without it projection will be created for only newly inserted data):

```
ALTER TABLE uk_price_paid
  MATERIALIZE PROJECTION projection_by_year_district_town
SETTINGS mutations_sync = 1;
```

## Test Performance

Let's run the same 3 queries.

Enable projections for selects:

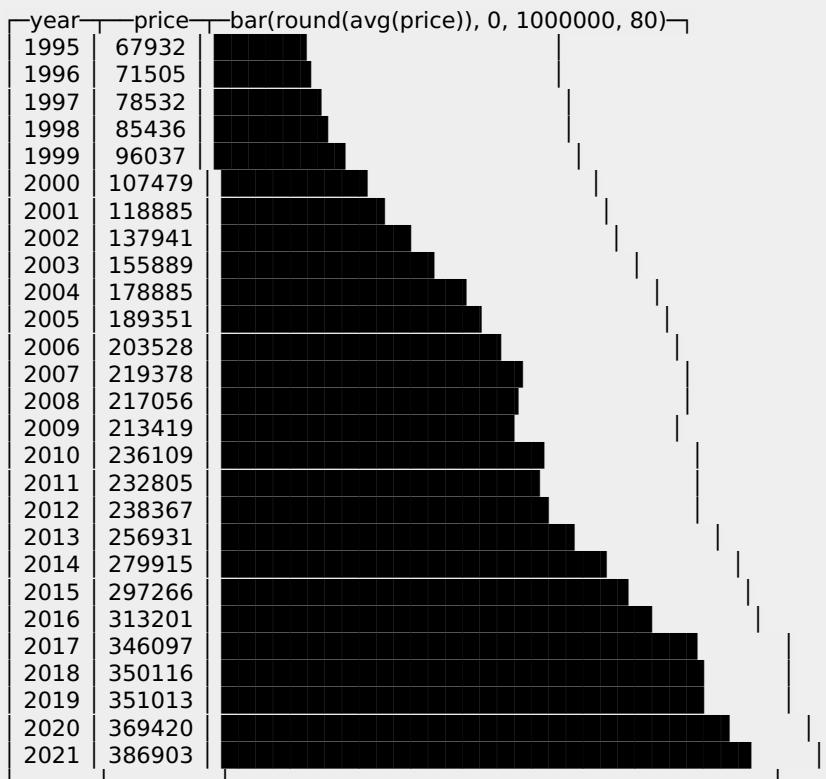
```
SET allow_experimental_projection_optimization = 1;
```

### Query 1. Average Price Per Year

Query:

```
SELECT
  toYear(date) AS year,
  round(avg(price)) AS price,
  bar(price, 0, 1000000, 80)
FROM uk_price_paid
GROUP BY year
ORDER BY year ASC;
```

Result:

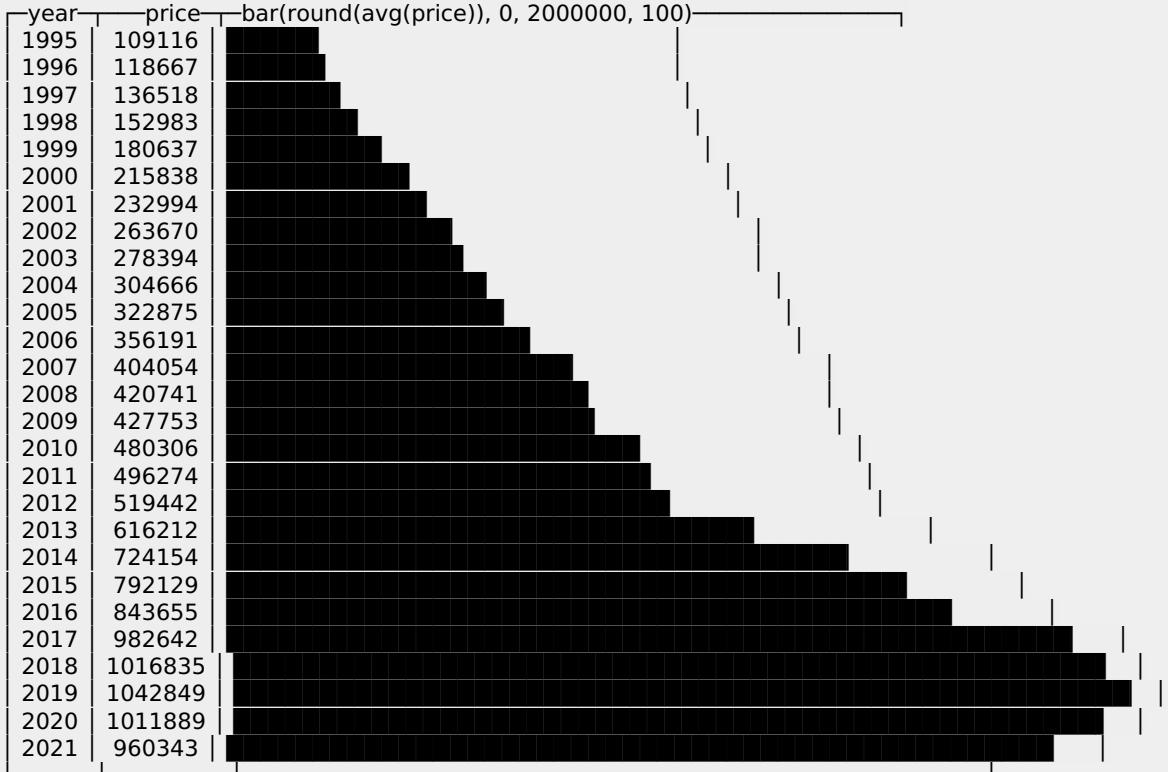


## Query 2. Average Price Per Year in London

Query:

```
SELECT
    toYear(date) AS year,
    round(avg(price)) AS price,
    bar(price, 0, 2000000, 100)
FROM uk_price_paid
WHERE town = 'LONDON'
GROUP BY year
ORDER BY year ASC;
```

Result:



### Query 3. The Most Expensive Neighborhoods

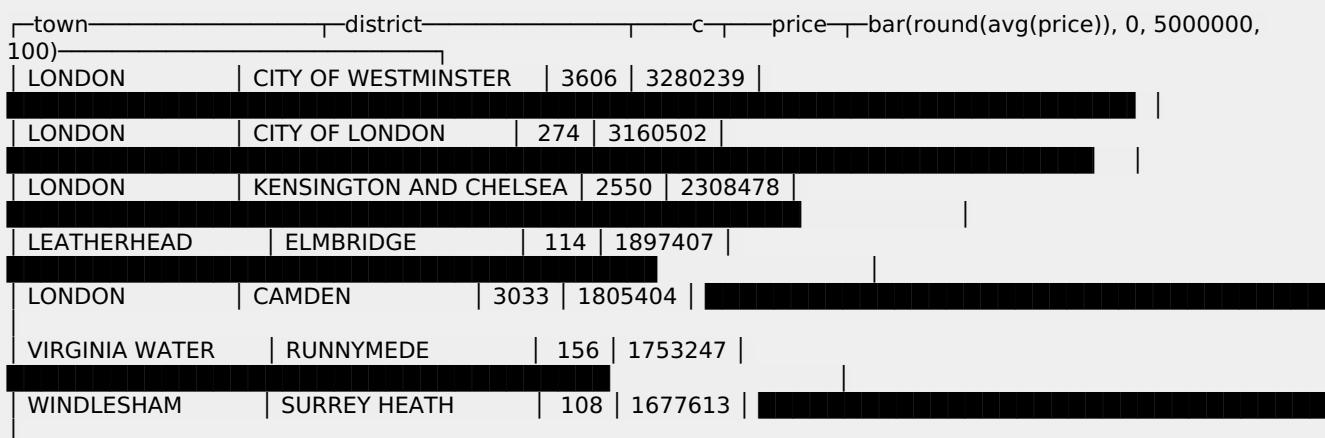
The condition (date >= '2020-01-01') needs to be modified to match projection dimension (toYear(date) >= 2020).

Query:

```

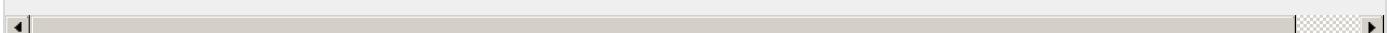
SELECT
  town,
  district,
  count() AS c,
  round(avg(price)) AS price,
  bar(price, 0, 5000000, 100)
FROM uk_price_paid
WHERE toYear(date) >= 2020
GROUP BY
  town,
  district
HAVING c >= 100
ORDER BY price DESC
LIMIT 100;
    
```

Result:



THORNTON HEATH	CROYDON	546   1671721   [REDACTED]
BARNET	ENFIELD	124   1505840   [REDACTED]
COBHAM	ELMBRIDGE	387   1237250   [REDACTED]
LONDON	ISLINGTON	2668   1236980   [REDACTED]
OXFORD	SOUTH OXFORDSHIRE	321   1220907   [REDACTED]
LONDON	RICHMOND UPON THAMES	704   1215551   [REDACTED]
LONDON	HOUNSLOW	671   1207493   [REDACTED]
ASCOT	WINDSOR AND MAIDENHEAD	407   1183299   [REDACTED]
BEACONSFIELD	BUCKINGHAMSHIRE	330   1175615   [REDACTED]
RICHMOND	RICHMOND UPON THAMES	874   1110444   [REDACTED]
LONDON	HAMMERSMITH AND FULHAM	3086   1053983   [REDACTED]
SURBITON	ELMBRIDGE	100   1011800   [REDACTED]
RADLETT	HERTSMERE	283   1011712   [REDACTED]
SALCOMBE	SOUTH HAMS	127   1011624   [REDACTED]
WEYBRIDGE	ELMBRIDGE	655   1007265   [REDACTED]
ESHER LEATHERHEAD	ELMBRIDGE   GUILDFORD	485   986581   [REDACTED]   202   977320   [REDACTED]
BURFORD	WEST OXFORDSHIRE	111   966893   [REDACTED]
BROCKENHURST	NEW FOREST	129   956675   [REDACTED]
HINDHEAD	WAVERLEY	137   953753   [REDACTED]
GERRARDS CROSS	BUCKINGHAMSHIRE	419   951121   [REDACTED]
EAST MOLESEY	ELMBRIDGE	192   936769   [REDACTED]
CHALFONT ST GILES	BUCKINGHAMSHIRE	146   925515   [REDACTED]
LONDON	TOWER HAMLETS	4388   918304   [REDACTED]
OLNEY	MILTON KEYNES	235   910646   [REDACTED]
HENLEY-ON-THAMES	SOUTH OXFORDSHIRE	540   902418   [REDACTED]
LONDON	SOUTHWARK	3885   892997   [REDACTED]
KINGSTON UPON THAMES	KINGSTON UPON THAMES	960   885969   [REDACTED]
LONDON CRANBROOK	EALING   TUNBRIDGE WELLS	2658   871755   [REDACTED]   431   862348   [REDACTED]
LONDON BELVEDERE	MERTON   BEXLEY	2099   859118   [REDACTED]   346   842423   [REDACTED]
GUILDFORD	WAVERLEY	143   841277   [REDACTED]
HARPENDEN	ST ALBANS	657   841216   [REDACTED]
LONDON	HACKNEY	3307   837090   [REDACTED]
LONDON	WANDSWORTH	6566   832663   [REDACTED]
MAIDENHEAD	BUCKINGHAMSHIRE	123   824299   [REDACTED]
KINGS LANGLEY	DACORUM	145   821331   [REDACTED]
BERKHAMSTED	DACORUM	543   818415   [REDACTED]
GREAT MISSENDEN	BUCKINGHAMSHIRE	226   802807   [REDACTED]
BILLINGSHURST	CHICHESTER	144   797829   [REDACTED]
WOKING STOCKBRIDGE	GUILDFORD   TEST VALLEY	176   793494   [REDACTED]   178   793269   [REDACTED]

EPSOM	REIGATE AND BANSTEAD	172   791862   [REDACTED]
TONBRIDGE	TUNBRIDGE WELLS	360   787876   [REDACTED]
TEDDINGTON	RICHMOND UPON THAMES	595   786492   [REDACTED]
TWICKENHAM	RICHMOND UPON THAMES	1155   786193   [REDACTED]
LYNDHURST	NEW FOREST	102   785593   [REDACTED]
LONDON	LAMBETH	5228   774574   [REDACTED]
LONDON	BARNET	3955   773259   [REDACTED]
OXFORD	VALE OF WHITE HORSE	353   772088   [REDACTED]
TONBRIDGE	MAIDSTONE	305   770740   [REDACTED]
LUTTERWORTH	HARBOROUGH	538   768634   [REDACTED]
WOODSTOCK	WEST OXFORDSHIRE	140   766037   [REDACTED]
MIDHURST	CHICHESTER	257   764815   [REDACTED]
MARLOW	BUCKINGHAMSHIRE	327   761876   [REDACTED]
LONDON	NEWHAM	3237   761784   [REDACTED]
ALDERLEY EDGE	CHESHIRE EAST	178   757318   [REDACTED]
LUTON	CENTRAL BEDFORDSHIRE	212   754283   [REDACTED]
PETWORTH	CHICHESTER	154   754220   [REDACTED]
ALRESFORD	WINCHESTER	219   752718   [REDACTED]
POTTERS BAR	WELWYN HATFIELD	174   748465   [REDACTED]
HASLEMERE	CHICHESTER	128   746907   [REDACTED]
TADWORTH	REIGATE AND BANSTEAD	502   743252   [REDACTED]
THAMES DITTON	ELMBRIDGE	244   741913   [REDACTED]
REIGATE	REIGATE AND BANSTEAD	581   738198   [REDACTED]
BOURNE END	BUCKINGHAMSHIRE	138   735190   [REDACTED]
SEVENOAKS	SEVENOAKS	1156   730018   [REDACTED]
OXTED	TANDRIDGE	336   729123   [REDACTED]
INGATESTONE	BRENTWOOD	166   728103   [REDACTED]
LONDON	BRENT	2079   720605   [REDACTED]
LONDON	HARINGEY	3216   717780   [REDACTED]
PURLEY	CROYDON	575   716108   [REDACTED]
WELWYN	WELWYN HATFIELD	222   710603   [REDACTED]
RICKMANSWORTH	THREE RIVERS	798   704571   [REDACTED]
BANSTEAD	REIGATE AND BANSTEAD	401   701293   [REDACTED]
CHIGWELL	EPPING FOREST	261   701203   [REDACTED]
PINNER	HARROW	528   698885   [REDACTED]
HASLEMERE	WAVERLEY	280   696659   [REDACTED]
SLOUGH	BUCKINGHAMSHIRE	396   694917   [REDACTED]
WALTON-ON-THAMES	ELMBRIDGE	946   692395   [REDACTED]
READING	SOUTH OXFORDSHIRE	318   691988   [REDACTED]
NORTHWOOD	HILLINGDON	271   690643   [REDACTED]
FELTHAM	HOUNSLAW	763   688595   [REDACTED]
ASHTead	MOLE VALLEY	303   687923   [REDACTED]
BARNET	BARNET	975   686980   [REDACTED]
WOKING	SURREY HEATH	283   686669   [REDACTED]
MALMESBURY	WILTSHIRE	323   683324   [REDACTED]
AMERSHAM	BUCKINGHAMSHIRE	496   680962   [REDACTED]
CHISLEHURST	BROMLEY	430   680209   [REDACTED]
HYTHE	FOLKESTONE AND HYTHE	490   676908   [REDACTED]
MAYFIELD	WEALDEN	101   676210   [REDACTED]
ASCOT	BRACKNELL FOREST	168   676004   [REDACTED]



## Summary

All 3 queries work much faster and read fewer rows.

### Query 1

```
no projection: 27 rows in set. Elapsed: 0.158 sec. Processed 26.32 million rows, 157.93 MB (166.57 million rows/s., 999.39 MB/s.)  
projection: 27 rows in set. Elapsed: 0.007 sec. Processed 105.96 thousand rows, 3.33 MB (14.58 million rows/s., 458.13 MB/s.)
```

### Query 2

```
no projection: 27 rows in set. Elapsed: 0.163 sec. Processed 26.32 million rows, 80.01 MB (161.75 million rows/s., 491.64 MB/s.)  
projection: 27 rows in set. Elapsed: 0.008 sec. Processed 105.96 thousand rows, 3.67 MB (13.29 million rows/s., 459.89 MB/s.)
```

### Query 3

```
no projection: 100 rows in set. Elapsed: 0.069 sec. Processed 26.32 million rows, 62.47 MB (382.13 million rows/s., 906.93 MB/s.)  
projection: 100 rows in set. Elapsed: 0.029 sec. Processed 8.08 thousand rows, 511.08 KB (276.06 thousand rows/s., 17.47 MB/s.)
```

## Test It in Playground

The dataset is also available in the [Online Playground](#).

## Cell Towers

This dataset is from [OpenCellid](#) - The world's largest Open Database of Cell Towers.

As of 2021, it contains more than 40 million records about cell towers (GSM, LTE, UMTS, etc.) around the world with their geographical coordinates and metadata (country code, network, etc).

OpenCellID Project is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License, and we redistribute a snapshot of this dataset under the terms of the same license. The up-to-date version of the dataset is available to download after sign in.

## Get the Dataset

1. Download the snapshot of the dataset from February 2021:  
[\[https://datasets.clickhouse.com/cell\\_towers.csv.xz\]](https://datasets.clickhouse.com/cell_towers.csv.xz) (729 MB).
2. Validate the integrity (optional step):

```
md5sum cell_towers.csv.xz  
8cf986f4a0d9f12c6f384a0e9192c908 cell_towers.csv.xz
```

3. Decompress it with the following command:

```
xz -d cell_towers.csv.xz
```

4. Create a table:

```

CREATE TABLE cell_towers
(
    radio Enum8("0" = 0, 'CDMA' = 1, 'GSM' = 2, 'LTE' = 3, 'NR' = 4, 'UMTS' = 5),
    mcc UInt16,
    net UInt16,
    area UInt16,
    cell UInt64,
    unit Int16,
    lon Float64,
    lat Float64,
    range UInt32,
    samples UInt32,
    changeable UInt8,
    created DateTime,
    updated DateTime,
    averageSignal UInt8
)
ENGINE = MergeTree ORDER BY (radio, mcc, net, created);

```

## 5. Insert the dataset:

```
clickhouse-client --query "INSERT INTO cell_towers FORMAT CSVWithNames" < cell_towers.csv
```

## Examples

### 1. A number of cell towers by type:

```
SELECT radio, count() AS c FROM cell_towers GROUP BY radio ORDER BY c DESC
```

radio	c
UMTS	20686487
LTE	12101148
GSM	9931312
CDMA	556344
NR	867

5 rows in set. Elapsed: 0.011 sec. Processed 43.28 million rows, 43.28 MB (3.83 billion rows/s., 3.83 GB/s.)

### 2. Cell towers by [mobile country code \(MCC\)](#):

```
SELECT mcc, count() FROM cell_towers GROUP BY mcc ORDER BY count() DESC LIMIT 10
```

mcc	count()
310	5024650
262	2622423
250	1953176
208	1891187
724	1836150
404	1729151
234	1618924
510	1353998
440	1343355
311	1332798

10 rows in set. Elapsed: 0.019 sec. Processed 43.28 million rows, 86.55 MB (2.33 billion rows/s., 4.65 GB/s.)

So, the top countries are: the USA, Germany, and Russia.

You may want to create an [External Dictionary](#) in ClickHouse to decode these values.

## Use case

Using `pointInPolygon` function.

1. Create a table where we will store polygons:

```
CREATE TEMPORARY TABLE moscow (polygon Array(Tuple(Float64, Float64)));
```

2. This is a rough shape of Moscow (without "new Moscow"):

```
INSERT INTO moscow VALUES ([(-37.84172564285271, 55.78000432402266), (-37.8381207618713, 55.775874525970494), (-37.83979446823122, 55.775626746008065), (-37.84243326983639, 55.77446586811748), (-37.84262672750849, 55.771974101091104), (-37.84153238623039, 55.77114545193181), (-37.841124690460184, 55.76722010265554), (-37.84239076983644, 55.76654891107098), (-37.842283558197025, 55.76258709833121), (-37.8421759312134, 55.758073999993734), (-37.84198330422974, 55.75381499999371), (-37.8416827275085, 55.749277102484484), (-37.84157576190186, 55.74794544108413), (-37.83897929098507, 55.74525257875241), (-37.83739676451868, 55.74404373042019), (-37.838732481460525, 55.74298009816793), (-37.841183997352545, 55.743060321833575), (-37.84097476190185, 55.73938799999373), (-37.84048155819702, 55.73570799999372), (-37.840095812164286, 55.73228210777237), (-37.83983814285274, 55.73080491981639), (-37.838636380279524, 55.72859509486539), (-37.8395161005249, 55.727705075632784), (-37.83897964285276, 55.722727886185154), (-37.83862557539366, 55.72034817326636), (-37.83559735744853, 55.71944437307499), (-37.835370708803126, 55.71831419154461), (-37.83738169402022, 55.71765218986692), (-37.83823396494291, 55.71691750159089), (-37.838056931213345, 55.71547311301385), (-37.836812846557606, 55.71221445615604), (-37.83522525396725, 55.709331054395555), (-37.83269301586908, 55.70953687463627), (-37.829667367706236, 55.70903403789297), (-37.83311126588435, 55.70552351822608), (-37.83058993121339, 55.70041317726053), (-37.82983872750851, 55.69883771404813), (-37.82934501586913, 55.69718947487017), (-37.828926414016685, 55.69504441658371), (-37.82876530422971, 55.69287499999378), (-37.82894754100031, 55.690759754047335), (-37.827697554878185, 55.68951421135665), (-37.82447346292115, 55.68965045405069), (-37.83136543914793, 55.68322046195302), (-37.833554015869154, 55.67814012759211), (-37.83544184655761, 55.67295011628339), (-37.83748038885474, 55.6672498719639), (-37.838960677246064, 55.66316274139358), (-37.83926093121332, 55.66046999999383), (-37.839025050262435, 55.65869897264431), (-37.83670784390257, 55.65794084879904), (-37.835656529083245, 55.65694309303843), (-37.83704060449217, 55.65689306460552), (-37.83696819873806, 55.65550363526252), (-37.83760389616388, 55.65487847246661), (-37.83687972750851, 55.65356745541324), (-37.83515216004943, 55.65155951234079), (-37.83312418518067, 55.64979413590619), (-37.82801726983639, 55.64640836412121), (-37.820614174591, 55.64164525405531), (-37.818908190475426, 55.6421883258084), (-37.81717543386075, 55.64112490388471), (-37.81690987037274, 55.63916106913107), (-37.815099354492155, 55.637925371757085), (-37.808769150787356, 55.633798276884455), (-37.80100123544311, 55.62873670012244), (-37.79598013491824, 55.62554336109055), (-37.78634567724606, 55.62033499605651), (-37.78334147619623, 55.618768681480326), (-37.77746201055901, 55.619855533402706), (-37.77527329626457, 55.61909966711279), (-37.77801986242668, 55.618770300976294), (-37.778212973541216, 55.617257701952106), (-37.77784818518065, 55.61574504433011), (-37.77016867724609, 55.61148576294007), (-37.760191219573976, 55.60599579539028), (-37.75338926983641, 55.60227892751446), (-37.746329965606634, 55.59920577639331), (-37.73939925396728, 55.59631430313617), (-37.73273665739439, 55.5935318803559), (-37.7299954450912, 55.59350760316188), (-37.7268679946899, 55.59469840523759), (-37.72626726983634, 55.59229549697373), (-37.7262673598022, 55.59081598950582), (-37.71897193121335, 55.5877595845419), (-37.70871550793456, 55.58393177431724), (-37.700497489410374, 55.580917323756644), (-37.69204305026244, 55.57778089778455), (-37.68544477378839, 55.57815154690915), (-37.68391050793454, 55.57472945079756), (-37.678803592590306, 55.57328235936491), (-37.6743402539673, 55.57255251445782), (-37.66813862698363, 55.57216388774464), (-37.617927457672096, 55.57505691895805), (-37.60443099999999, 55.5757737568051), (-37.599683515869145, 55.57749105910326), (-37.59754177842709, 55.57796291823627), (-37.59625834786988, 55.57906686095235), (-37.59501783265684, 55.57746616444403), (-37.593090671936025, 55.57671634534502), (-37.587018007904, 55.577944600233785), (-37.578692203704804, 55.57982895000019), (-37.57327546607398, 55.58116294118248), (-37.57385012109279, 55.581550362779), (-37.57399562266922, 55.5820107079112), (-37.5735356072979, 55.58226289171689), (-37.57290393054962, 55.582393529795155), (-37.57037722355653, 55.581919415056234), (-37.5592298306885, 55.584471614867844), (-37.54189249206543, 55.58867650795186), (-37.5297256269836, 55.59158133551745), (-37.517837865081766, 55.59443656218868), (-37.51200186508174, 55.59635625174229), (-37.506808949737554, 55.59907823904434), (-37.49820432275389, 55.6062944994944), (-37.494406071441674, 55.60967103463367), (-37.494760001358024, 55.61066689753365), (-37.49397137107085, 55.61220931698269), (-37.49016528606031, 55.613417718449064), (-37.48773249206542, 55.61530616333343), (-37.47921386508177, 55.622640129112334), (-37.470652153442394, 55.62993723476164), (-37.46273446298218, 55.6368075123157), (-37.46350692265317, 55.64068225239439), (-37.46050283203121, 55.640794546982576), (-37.457627470916734, 55.64118904154646), (-37.450718034393326, 55.64690488145138), (-37.44239252645875, 55.65397824729769), (-37.434587576721185, 55.66053543155961), (-37.43582144975277, 55.661693766520735), (-37.43576786245721, 55.662755031737014), (-37.430982915344174, 55.664610641628116), (-37.428547447097685, 55.66778515273695), (-37.42945134592044, 55.668633314343566), (-37.42859571562949, 55.66948145750025), (-37.4262836402282, 55.670813882451405), (-37.418709037048295, 55.6811141674414), (-37.41922139651101, 55.68235377885389), (-37.419218771842885, 55.68359335082235), (-37.417196501327446, 55.684375235224735), (-37.41607020370478, 55.68540557585352), (-37.415640857147146, 55.68686637150793), (-37.414632153442334, 55.68903015131686), (-37.413344899475064, 55.690896881757396), (-37.41171432275391, 55.69264232162232), (-37.40948282275393, 55.69455101638112), (-37.40703674603271, 55.69638690385348), (-37.39607169577025, 55.70451821283731), (-37.38952706878662, 55.70942491932811), (-37.387778313491815, 55.71149057784176), (-37.39049275399779, 55.71419814298992), (-37.385557272491454, 55.7155489617061), (-37.38388335714726, 55.71849856042102), (-37.378368238098155, 55.7292763261685), (-37.37763597123337, 55.730845879211614), (-37.37890062088197, 55.73167906388319), (-37.37750451918789, 55.734703664681774), (-37.375610832015965, 55.734851959522246), (-37.3723813571472, 55.74105626086403), (-37.37014935714723, 55.746115620904355), (-37.36944173016362, 55.750883999993725), (-37.36975304365541, 55.76335905525834), (-37.37244070571134, 55.76432079697595), (-37.3724259757175, 55.76636979670426), (-37.37244070571134, 55.76432079697595), (-37.3724259757175, 55.76636979670426)
```

```
(3/.369922155/5/884, 55./6/3541/953104), (3/.369892695//02/5, 55./68234193165/5), (3/.3/0214/301635/5,
55.782312184391266), (37.370493611114505, 55.78436801120489), (37.37120164550783, 55.78596427165359),
(37.37284851456452, 55.7874378183096), (37.37608325135799, 55.7886695054807), (37.3764587460632,
55.78947647305964), (37.3753000265506, 55.79146512926804), (37.38235915344241, 55.79899647809345),
(37.384344043655396, 55.80113596939471), (37.38594269577028, 55.80322699999366), (37.38711208598329,
55.804919036911976), (37.3880239841309, 55.806610999993666), (37.38928977249147, 55.81001864976979),
(37.39038389947512, 55.81348641242801), (37.39235781481933, 55.81983538336746), (37.393709457672124,
55.82417822811877), (37.394685720901464, 55.82792275755836), (37.39557615344238, 55.830447148154136),
(37.39844478226658, 55.83167107969975), (37.40019761214057, 55.83151823557964), (37.400398790382326,
55.83264967594742), (37.39659544313046, 55.83322180909622), (37.39667059524539, 55.83402792148566),
(37.39682089947515, 55.83638877400216), (37.39643489154053, 55.83861656112751), (37.3955338994751,
55.84072348043264), (37.392680272491454, 55.84502158126453), (37.39241188227847, 55.84659117913199),
(37.392529730163616, 55.84816071336481), (37.39486835714723, 55.85288092980303), (37.39873052645878,
55.859893456073635), (37.40272161111449, 55.86441833633205), (37.40697072750854, 55.867579567544375),
(37.410007082016016, 55.868369880337), (37.4120992989502, 55.86920843741314), (37.412668021163924,
55.87055369615854), (37.41482461111453, 55.87170587948249), (37.41862266137694, 55.873183961039565),
(37.42413732540892, 55.874879126654704), (37.4312182698669, 55.875614937236705), (37.43111093783558,
55.8762723478417), (37.43332105622856, 55.87706546369396), (37.43385747619623, 55.87790681284802),
(37.441303050262405, 55.88027084462084), (37.44747234260555, 55.87942070143253), (37.44716141796871,
55.88072960917233), (37.44769797085568, 55.88121221323979), (37.45204320500181, 55.882080694420715),
(37.45673176190186, 55.882346110794586), (37.463383999999984, 55.88252729504517), (37.46682797486874,
55.88294937719063), (37.470014457672086, 55.88361266759345), (37.47751410450743, 55.88546991372396),
(37.47860317658232, 55.88534929207307), (37.48165826025772, 55.882563306475106), (37.48316434442331,
55.8815803226785), (37.483831555817645, 55.882427612793315), (37.483182967125686, 55.88372791409729),
(37.483092277908824, 55.88495581062434), (37.4855716508179, 55.8875561994203), (37.486440636245746,
55.887827444039566), (37.49014203439328, 55.88897899871799), (37.493210285705544, 55.890208937135604),
(37.497512451065035, 55.891342397444696), (37.49780744510645, 55.89174030252967), (37.49940333499519,
55.89239745507079), (37.50018383334346, 55.89339220941865), (37.52421672750851, 55.903869074155224),
(37.52977457672118, 55.90564076517974), (37.53503220370484, 55.90661661218259), (37.54042858064267,
55.90714113744566), (37.54320461007303, 55.905645048442985), (37.545686966066306, 55.906608607018505),
(37.54743976120755, 55.90788552162358), (37.55796999999999, 55.90901557907218), (37.572711542327866,
55.91059395704873), (37.57942799999999, 55.91073854155573), (37.58502865872187, 55.91009969268444),
(37.58739968913264, 55.90794809960554), (37.59131567193598, 55.908713267595054), (37.612687423278814,
55.902866854295375), (37.62348079629517, 55.90041967242986), (37.635797880950896, 55.898141151686396),
(37.649487626983664, 55.89639275532968), (37.65619302513125, 55.89572360207488), (37.66294133862307,
55.895295577183965), (37.66874564418033, 55.89505457604897), (37.67375601586915, 55.89254677027454),
(37.67744661901856, 55.8947775867987), (37.688347, 55.89450045676125), (37.69480554232789,
55.89422926332761), (37.70107096560668, 55.89322256101114), (37.705962965606716, 55.891763491662616),
(37.711885134918205, 55.889110234998974), (37.71682005026245, 55.886577568759876), (37.7199315476074,
55.88458159806678), (37.72234560316464, 55.882281005794134), (37.72364385977171, 55.8809452036196),
(37.725371142837474, 55.8809722706006), (37.727870902099546, 55.88037213862385), (37.73394330422971,
55.877941504088696), (37.745339592590376, 55.87208120378722), (37.75525267724611, 55.86703807949492),
(37.76919976190188, 55.859821640197474), (37.827835219574, 55.82962968399116), (37.8334143888553,
55.82575289922351), (37.83652584655761, 55.82188784027888), (37.83809213491821, 55.81612575504693),
(37.83605359521481, 55.81460347077685), (37.83632178569025, 55.81276696067908), (37.838623105812026,
55.811486181656385), (37.83912198147584, 55.807329380532785), (37.839079078033414, 55.80510270463816),
(37.83965844708251, 55.79940712529036), (37.840581150787344, 55.79131399999368), (37.84172564285271,
55.78000432402266));
```

### 3. Check how many cell towers are in Moscow:

```
SELECT count() FROM cell_towers WHERE pointInPolygon((lon, lat), (SELECT * FROM moscow))
```

```
count()
310463
```

1 rows in set. Elapsed: 0.067 sec. Processed 43.28 million rows, 692.42 MB (645.83 million rows/s., 10.33 GB/s.)

The data is also available for interactive queries in the [Playground](#), [example](#).

Although you cannot create temporary tables there.

## New York Public Library "What's on the Menu?" Dataset

The dataset is created by the New York Public Library. It contains historical data on the menus of hotels, restaurants and cafes with the dishes along with their prices.

Source: <http://menus.nypl.org/data>

The data is in public domain.

The data is from library's archive and it may be incomplete and difficult for statistical analysis.

Nevertheless it is also very yummy.

The size is just 1.3 million records about dishes in the menus — it's a very small data volume for ClickHouse, but it's still a good example.

## Download the Dataset

Run the command:

```
wget https://s3.amazonaws.com/menusdata.nypl.org/gzips/2021_08_01_07_01_17_data.tgz
```

Replace the link to the up to date link from <http://menus.nypl.org/data> if needed.

Download size is about 35 MB.

## Unpack the Dataset

```
tar xvf 2021_08_01_07_01_17_data.tgz
```

Uncompressed size is about 150 MB.

The data is normalized consisted of four tables:

- **Menu** — Information about menus: the name of the restaurant, the date when menu was seen, etc.
- **Dish** — Information about dishes: the name of the dish along with some characteristic.
- **MenuPage** — Information about the pages in the menus, because every page belongs to some menu.
- **MenuItem** — An item of the menu. A dish along with its price on some menu page: links to dish and menu page.

## Create the Tables

We use **Decimal** data type to store prices.

```

CREATE TABLE dish
(
    id UInt32,
    name String,
    description String,
    menus_appeared UInt32,
    times_appeared Int32,
    first_appeared UInt16,
    last_appeared UInt16,
    lowest_price Decimal64(3),
    highest_price Decimal64(3)
) ENGINE = MergeTree ORDER BY id;

CREATE TABLE menu
(
    id UInt32,
    name String,
    sponsor String,
    event String,
    venue String,
    place String,
    physical_description String,
    occasion String,
    notes String,
    call_number String,
    keywords String,
    language String,
    date String,
    location String,
    location_type String,
    currency String,
    currency_symbol String,
    status String,
    page_count UInt16,
    dish_count UInt16
) ENGINE = MergeTree ORDER BY id;

CREATE TABLE menu_page
(
    id UInt32,
    menu_id UInt32,
    page_number UInt16,
    image_id String,
    full_height UInt16,
    full_width UInt16,
    uuid UUID
) ENGINE = MergeTree ORDER BY id;

CREATE TABLE menu_item
(
    id UInt32,
    menu_page_id UInt32,
    price Decimal64(3),
    high_price Decimal64(3),
    dish_id UInt32,
    created_at DateTime,
    updated_at DateTime,
    xpos Float64,
    ypos Float64
) ENGINE = MergeTree ORDER BY id;

```

## Import the Data

Upload data into ClickHouse, run:

```
clickhouse-client --format_csv_allow_single_quotes 0 --input_format_null_as_default 0 --query "INSERT INTO dish
FORMAT CSVWithNames" < Dish.csv
clickhouse-client --format_csv_allow_single_quotes 0 --input_format_null_as_default 0 --query "INSERT INTO menu
FORMAT CSVWithNames" < Menu.csv
clickhouse-client --format_csv_allow_single_quotes 0 --input_format_null_as_default 0 --query "INSERT INTO
menu_page FORMAT CSVWithNames" < MenuPage.csv
clickhouse-client --format_csv_allow_single_quotes 0 --input_format_null_as_default 0 --date_time_input_format
best_effort --query "INSERT INTO menu_item FORMAT CSVWithNames" < MenuItem.csv
```

We use **CSVWithNames** format as the data is represented by CSV with header.

We disable **format\_csv\_allow\_single\_quotes** as only double quotes are used for data fields and single quotes can be inside the values and should not confuse the CSV parser.

We disable **input\_format\_null\_as\_default** as our data does not have **NULL**. Otherwise ClickHouse will try to parse \N sequences and can be confused with \ in data.

The setting **date\_time\_input\_format best\_effort** allows to parse **DateTime** fields in wide variety of formats. For example, ISO-8601 without seconds like '2000-01-01 01:02' will be recognized. Without this setting only fixed DateTime format is allowed.

## Denormalize the Data

Data is presented in multiple tables in **normalized form**. It means you have to perform **JOIN** if you want to query, e.g. dish names from menu items.

For typical analytical tasks it is way more efficient to deal with pre-JOINed data to avoid doing **JOIN** every time. It is called "denormalized" data.

We will create a table `menu_item_denorm` which will contain all the data JOINed together:

```

CREATE TABLE menu_item_denorm
ENGINE = MergeTree ORDER BY (dish_name, created_at)
AS SELECT
    price,
    high_price,
    created_at,
    updated_at,
    xpos,
    ypos,
    dish.id AS dish_id,
    dish.name AS dish_name,
    dish.description AS dish_description,
    dish.menus_appeared AS dish_menus_appeared,
    dish.times_appeared AS dish_times_appeared,
    dish.first_appeared AS dish_first_appeared,
    dish.last_appeared AS dish_last_appeared,
    dish.lowest_price AS dish_lowest_price,
    dish.highest_price AS dish_highest_price,
    menu.id AS menu_id,
    menu.name AS menu_name,
    menu.sponsor AS menu_sponsor,
    menu.event AS menu_event,
    menu.venue AS menu_venue,
    menu.place AS menu_place,
    menu.physical_description AS menu_physical_description,
    menu.occasion AS menu_occasion,
    menu.notes AS menu_notes,
    menu.call_number AS menu_call_number,
    menu.keywords AS menu_keywords,
    menu.language AS menu_language,
    menu.date AS menu_date,
    menu.location AS menu_location,
    menu.location_type AS menu_location_type,
    menu.currency AS menu_currency,
    menu.currency_symbol AS menu_currency_symbol,
    menu.status AS menu_status,
    menu.page_count AS menu_page_count,
    menu.dish_count AS menu_dish_count
FROM menu_item
JOIN dish ON menu_item.dish_id = dish.id
JOIN menu_page ON menu_item.menu_page_id = menu_page.id
JOIN menu ON menu_page.menu_id = menu.id;

```

## Validate the Data

Query:

```
SELECT count() FROM menu_item_denorm;
```

Result:

count()
1329175

## Run Some Queries

Averaged historical prices of dishes

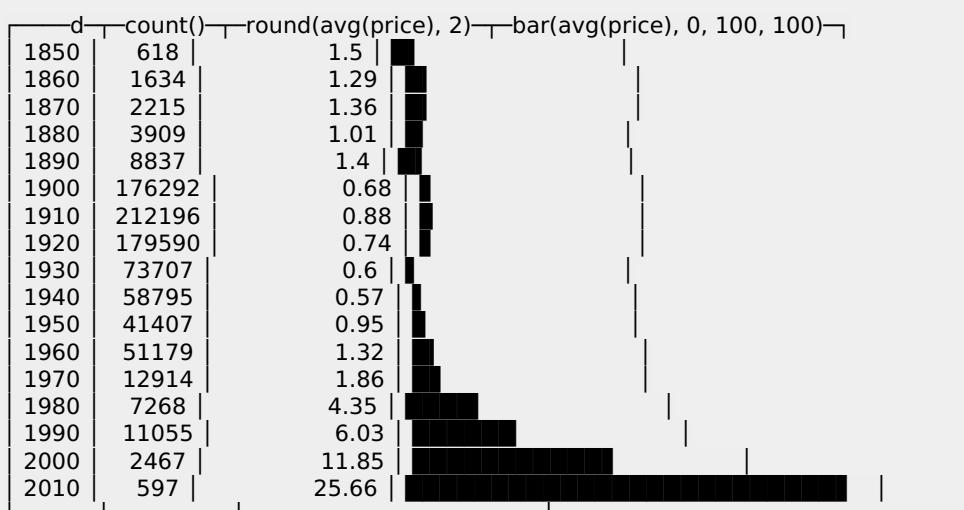
Query:

```

SELECT
    round(toUInt32OrZero(extract(menu_date, '^\\d{4}')), -1) AS d,
    count(),
    round(avg(price), 2),
    bar(avg(price), 0, 100, 100)
FROM menu_item_denorm
WHERE (menu_currency = 'Dollars') AND (d > 0) AND (d < 2022)
GROUP BY d
ORDER BY d ASC;

```

Result:



Take it with a grain of salt.

## Burger Prices

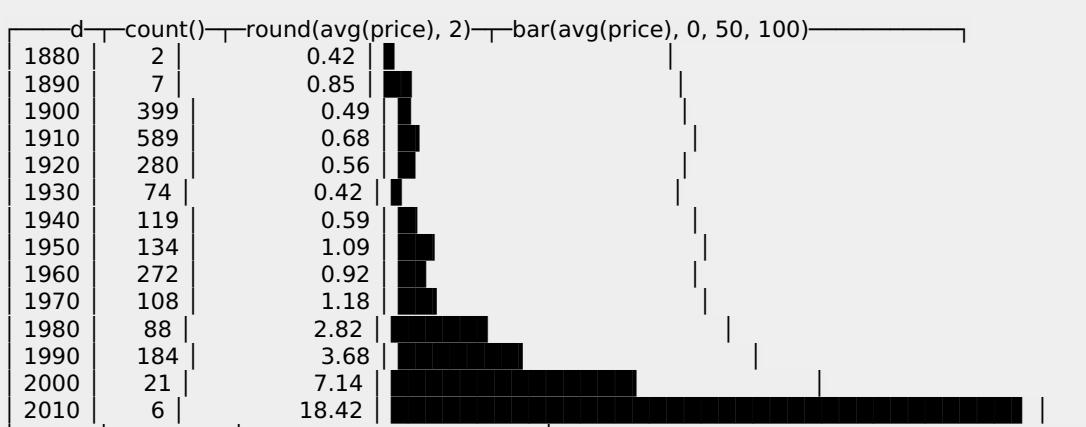
Query:

```

SELECT
    round(toUInt32OrZero(extract(menu_date, '^\\d{4}')), -1) AS d,
    count(),
    round(avg(price), 2),
    bar(avg(price), 0, 50, 100)
FROM menu_item_denorm
WHERE (menu_currency = 'Dollars') AND (d > 0) AND (d < 2022) AND (dish_name ILIKE '%burger%')
GROUP BY d
ORDER BY d ASC;

```

Result:

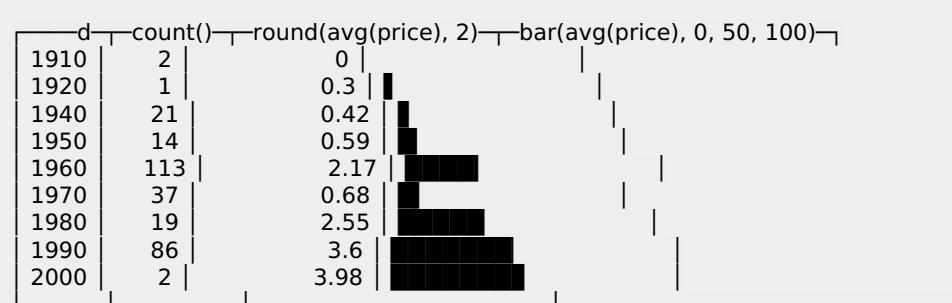


## Vodka

Query:

```
SELECT
    round(toUInt32OrZero(extract(menu_date, '^\\d{4}')), -1) AS d,
    count(),
    round(avg(price), 2),
    bar(avg(price), 0, 50, 100)
FROM menu_item_denorm
WHERE (menu_currency IN ('Dollars', '')) AND (d > 0) AND (d < 2022) AND (dish_name ILIKE '%vodka%')
GROUP BY d
ORDER BY d ASC;
```

Result:



To get vodka we have to write ILIKE '%vodka%' and this definitely makes a statement.

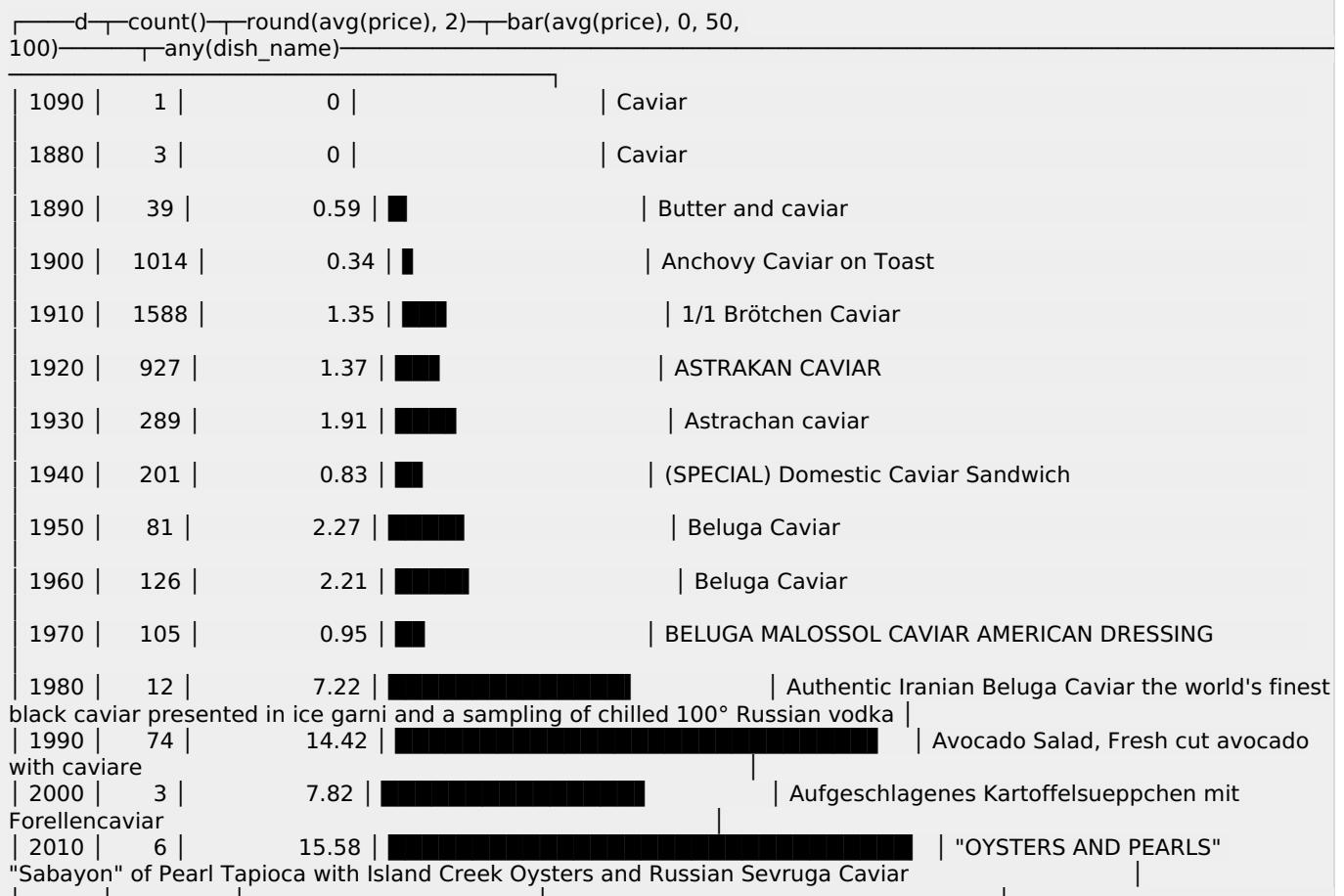
## Caviar

Let's print caviar prices. Also let's print a name of any dish with caviar.

Query:

```
SELECT
    round(toUInt32OrZero(extract(menu_date, '^\\d{4}')), -1) AS d,
    count(),
    round(avg(price), 2),
    bar(avg(price), 0, 50, 100),
    any(dish_name)
FROM menu_item_denorm
WHERE (menu_currency IN ('Dollars', '')) AND (d > 0) AND (d < 2022) AND (dish_name ILIKE '%caviar%')
GROUP BY d
ORDER BY d ASC;
```

Result:



At least they have caviar with vodka. Very nice.

## Online Playground

The data is uploaded to ClickHouse Playground, [example](#).

## OnTime

This dataset can be obtained in two ways:

- import from raw data
- download of prepared partitions

## Import from Raw Data

Downloading data:

```
echo
https://transtats.bts.gov/PREZIP/On_Time_Reportng_Carrier_On_Time_Performance_1987_present_{1987..2021}_{1..12}.zip | xargs -P10 wget --no-check-certificate --continue
```

Creating a table:

```
CREATE TABLE `ontime`
(
    `Year`          UInt16,
    `Quarter`       UInt8,
```

```

`Month`           UInt8,
`DayofMonth`     UInt8,
`DayOfWeek`      UInt8,
`FlightDate`     Date,
`Reporting_Airline` String,
`DOT_ID_Reported_Airline` Int32,
`IATA_CODE_Reported_Airline` String,
`Tail_Number`    String,
`Flight_Number_Reported_Airline` 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`          Nullable(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,
`DivActualElapsed` String,
`DivArrDelay`     String,
`DivDistance`    String,
`Div1Airport`     String,
`Div1AirportID`   Int32,
`Div1AirportSeqID` Int32,
`Div1WheelsOn`   String,
`Div1TotalGTime` String,
`Div1LongestGTime` String,
`Div1WheelsOff`  String,
`Div1TailNum`    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 (IATA_CODE_Reported_Airline, FlightDate)
SETTINGS index_granularity = 8192;

```

Loading data with multiple threads:

```
ls -1 *.zip | xargs -I{} -P $(nproc) bash -c "echo {}; unzip -cq {} '*.csv' | sed 's/\.\.00//g' | clickhouse-client --input_format_with_names_use_header=0 --query='INSERT INTO ontyme FORMAT CSVWithNames'"
```

(if you will have memory shortage or other issues on your server, remove the `-P $(nproc)` part)

## Download of Prepared Partitions

```
$ curl -O https://datasets.clickhouse.com/ontime/partitions/ontime.tar
$ tar xvf ontyme.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.ontime"
```

### Info

If you will run the queries described below, you have to use the full table name, `datasets.ontime`.

## Queries

Q0.

```
SELECT avg(c1)
FROM
(
  SELECT Year, Month, count(*) AS c1
  FROM ontime
  GROUP BY Year, Month
);
```

Q1. The number of flights per day from the year 2000 to 2008

```
SELECT DayOfWeek, count(*) AS c
FROM ontime
WHERE Year>=2000 AND Year<=2008
GROUP BY DayOfWeek
ORDER BY c DESC;
```

Q2. The number of flights delayed by more than 10 minutes, grouped by the day of the week, for 2000-2008

```
SELECT DayOfWeek, count(*) AS c
FROM ontime
WHERE DepDelay>10 AND Year>=2000 AND Year<=2008
GROUP BY DayOfWeek
ORDER BY c DESC;
```

Q3. The number of delays by the airport for 2000-2008

```
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. The number of delays by carrier for 2007

```
SELECT IATA_CODE_Reported_Airline AS Carrier, count(*)
FROM ontime
WHERE DepDelay>10 AND Year=2007
GROUP BY Carrier
ORDER BY count(*) DESC;
```

Q5. The percentage of delays by carrier for 2007

```

SELECT Carrier, c, c2, c*100/c2 as c3
FROM
(
  SELECT
    IATA_CODE_Reported_Airline AS Carrier,
    count(*) AS c
  FROM ontime
  WHERE DepDelay>10
    AND Year=2007
  GROUP BY Carrier
) q
JOIN
(
  SELECT
    IATA_CODE_Reported_Airline AS Carrier,
    count(*) AS c2
  FROM ontime
  WHERE Year=2007
  GROUP BY Carrier
) qq USING Carrier
ORDER BY c3 DESC;

```

Better version of the same query:

```

SELECT IATA_CODE_Reported_Airline AS Carrier, avg(DepDelay>10)*100 AS c3
FROM ontime
WHERE Year=2007
GROUP BY Carrier
ORDER BY c3 DESC

```

Q6. The previous request for a broader range of years, 2000-2008

```

SELECT Carrier, c, c2, c*100/c2 as c3
FROM
(
  SELECT
    IATA_CODE_Reported_Airline AS Carrier,
    count(*) AS c
  FROM ontime
  WHERE DepDelay>10
    AND Year>=2000 AND Year<=2008
  GROUP BY Carrier
) q
JOIN
(
  SELECT
    IATA_CODE_Reported_Airline AS Carrier,
    count(*) AS c2
  FROM ontime
  WHERE Year>=2000 AND Year<=2008
  GROUP BY Carrier
) qq USING Carrier
ORDER BY c3 DESC;

```

Better version of the same query:

```

SELECT IATA_CODE_Reported_Airline AS Carrier, avg(DepDelay>10)*100 AS c3
FROM ontime
WHERE Year>=2000 AND Year<=2008
GROUP BY Carrier
ORDER BY c3 DESC;

```

Q7. Percentage of flights delayed for more than 10 minutes, by year

```

SELECT Year, c1/c2
FROM
(
  select
    Year,
    count(*)*100 as c1
  from ontime
  WHERE DepDelay>10
  GROUP BY Year
) q
JOIN
(
  select
    Year,
    count(*) as c2
  from ontime
  GROUP BY Year
) qq USING (Year)
ORDER BY Year;

```

Better version of the same query:

```

SELECT Year, avg(DepDelay>10)*100
FROM ontime
GROUP BY Year
ORDER BY Year;

```

Q8. The most popular destinations by the number of directly connected cities for various year ranges

```

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), IATA_CODE_Reported_Airline AS 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;

```

Bonus:

```

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;

```

You can also play with the data in Playground, [example](#).

This performance test was created by Vadim Tkachenko. See:

- <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>

## Installation

### System Requirements

ClickHouse can run on any Linux, FreeBSD, or Mac OS X with x86\_64, AArch64, or PowerPC64LE CPU architecture.

Official pre-built binaries are typically compiled for x86\_64 and leverage SSE 4.2 instruction set, so unless otherwise stated usage of CPU that supports it becomes an additional system requirement. Here's the command to check if current CPU has support for SSE 4.2:

```
$ grep -q sse4_2 /proc/cpuinfo && echo "SSE 4.2 supported" || echo "SSE 4.2 not supported"
```

To run ClickHouse on processors that do not support SSE 4.2 or have AArch64 or PowerPC64LE architecture, you should [build ClickHouse from sources](#) with proper configuration adjustments.

## Available Installation Options

### From DEB Packages

It is recommended to use official pre-compiled `deb` packages for Debian or Ubuntu. Run these commands to install packages:

```
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.com/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
```

You can replace `stable` with `its` or `testing` to use different [release trains](#) based on your needs.

You can also download and install packages manually from [here](#).

#### Packages

- `clickhouse-common-static` — Installs ClickHouse compiled binary files.
- `clickhouse-server` — Creates a symbolic link for `clickhouse-server` and installs the default server configuration.
- `clickhouse-client` — Creates a symbolic link for `clickhouse-client` and other client-related tools. and installs client configuration files.
- `clickhouse-common-static-dbg` — Installs ClickHouse compiled binary files with debug info.

### Attention

If you need to install specific version of ClickHouse you have to install all packages with the same version:

```
sudo apt-get install clickhouse-server=21.8.5.7 clickhouse-client=21.8.5.7 clickhouse-common-static=21.8.5.7
```

### From RPM Packages

It is recommended to use official pre-compiled `rpm` packages for CentOS, RedHat, and all other rpm-based Linux distributions.

First, you need to add the official repository:

```
sudo yum install yum-utils  
sudo rpm --import https://repo.clickhouse.com/CClickHOUSE-KEY.GPG  
sudo yum-config-manager --add-repo https://repo.clickhouse.com/rpm/stable/x86_64
```

If you want to use the most recent version, replace `stable` with `testing` (this is recommended for your testing environments). `prestable` is sometimes also available.

Then run these commands to install packages:

```
sudo yum install clickhouse-server clickhouse-client
```

You can also download and install packages manually from [here](#).

## From Tgz Archives

It is recommended to use official pre-compiled tgz archives for all Linux distributions, where installation of deb or rpm packages is not possible.

The required version can be downloaded with curl or wget from repository <https://repo.clickhouse.com/tgz/>. After that downloaded archives should be unpacked and installed with installation scripts. Example for the latest version:

```
export LATEST_VERSION=`curl https://api.github.com/repos/ClickHouse/ClickHouse/tags 2>/dev/null | grep -E '[0-9]+\.[0-9]+\.[0-9]+[.\[0-9]]+' | head -n 1`  
curl -O https://repo.clickhouse.com/tgz/clickhouse-common-static-$LATEST_VERSION.tgz  
curl -O https://repo.clickhouse.com/tgz/clickhouse-common-static-dbg-$LATEST_VERSION.tgz  
curl -O https://repo.clickhouse.com/tgz/clickhouse-server-$LATEST_VERSION.tgz  
curl -O https://repo.clickhouse.com/tgz/clickhouse-client-$LATEST_VERSION.tgz  
  
tar -xzvf clickhouse-common-static-$LATEST_VERSION.tgz  
sudo clickhouse-common-static-$LATEST_VERSION/install/doinst.sh  
  
tar -xzvf clickhouse-common-static-dbg-$LATEST_VERSION.tgz  
sudo clickhouse-common-static-dbg-$LATEST_VERSION/install/doinst.sh  
  
tar -xzvf clickhouse-server-$LATEST_VERSION.tgz  
sudo clickhouse-server-$LATEST_VERSION/install/doinst.sh  
sudo /etc/init.d/clickhouse-server start  
  
tar -xzvf clickhouse-client-$LATEST_VERSION.tgz  
sudo clickhouse-client-$LATEST_VERSION/install/doinst.sh
```

For production environments, it's recommended to use the latest stable-version. You can find its number on GitHub page <https://github.com/ClickHouse/ClickHouse/tags> with postfix -stable.

## From Docker Image

To run ClickHouse inside Docker follow the guide on [Docker Hub](#). Those images use official deb packages inside.

## Single Binary

You can install ClickHouse on Linux using a single portable binary from the latest commit of the master branch: [<https://builds.clickhouse.com/master/amd64/clickhouse>].

```
curl -O 'https://builds.clickhouse.com/master/amd64/clickhouse' && chmod a+x clickhouse  
sudo ./clickhouse install
```

## From Precompiled Binaries for Non-Standard Environments

For non-Linux operating systems and for AArch64 CPU architecture, ClickHouse builds are provided as a cross-compiled binary from the latest commit of the master branch (with a few hours delay).

- [MacOS x86\\_64](#) — curl -O 'https://builds.clickhouse.com/master/macos/clickhouse' && chmod a+x ./clickhouse
- [MacOS Aarch64 \(Apple Silicon\)](#) — curl -O 'https://builds.clickhouse.com/master/macos-aarch64/clickhouse' && chmod a+x ./clickhouse
- [FreeBSD x86\\_64](#) — curl -O 'https://builds.clickhouse.com/master/freebsd/clickhouse' && chmod a+x ./clickhouse

- **Linux AArch64** — `curl -O 'https://builds.clickhouse.com/master/aarch64/clickhouse' && chmod a+x ./clickhouse`

After downloading, you can use the `clickhouse` client to connect to the server, or `clickhouse local` to process local data.

Run `sudo ./clickhouse install` if you want to install clickhouse system-wide (also with needed configuration files, configuring users etc.). After that run `clickhouse start` commands to start the clickhouse-server and `clickhouse-client` to connect to it.

These builds are not recommended for use in production environments because they are less thoroughly tested, but you can do so on your own risk. They also have only a subset of ClickHouse features available.

## From Sources

To manually compile ClickHouse, follow the instructions for [Linux](#) or [Mac OS X](#).

You can compile packages and install them or use programs without installing packages. Also by building manually you can disable SSE 4.2 requirement or build for AArch64 CPUs.

```
Client: programs/clickhouse-client  
Server: programs/clickhouse-server
```

You'll need to create a data and metadata folders and `chown` them for the desired user. Their paths can be changed in server config (`src/programs/server/config.xml`), by default they are:

```
/var/lib/clickhouse/data/default/  
/var/lib/clickhouse/metadata/default/
```

On Gentoo, you can just use `emerge clickhouse` to install ClickHouse from sources.

## Launch

To start the server as a daemon, run:

```
$ sudo service clickhouse-server start
```

If you do not have `service` command, run as

```
$ sudo /etc/init.d/clickhouse-server start
```

See the logs in the `/var/log/clickhouse-server/` directory.

If the server does not start, check the configurations in the file `/etc/clickhouse-server/config.xml`.

You can also manually launch the server from the console:

```
$ clickhouse-server --config-file=/etc/clickhouse-server/config.xml
```

In this case, the log will be printed to the console, which is convenient during development.

If the configuration file is in the current directory, you do not need to specify the `--config-file` parameter. By default, it uses `./config.xml`.

ClickHouse supports access restriction settings. They are located in the `users.xml` file (next to `config.xml`). By default, access is allowed from anywhere for the `default` user, without a password. See `user/default/networks`.

For more information, see the section "[Configuration Files](#)".

After launching server, you can use the command-line client to connect to it:

```
$ clickhouse-client
```

By default, it connects to localhost:9000 on behalf of the user default without a password. It can also be used to connect to a remote server using `--host` argument.

The terminal must use UTF-8 encoding.

For more information, see the section “[Command-line client](#)”.

Example:

```
$ ./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.

:)

## Congratulations, the system works!

To continue experimenting, you can download one of the test data sets or go through [tutorial](#).

# ClickHouse Tutorial

## What to Expect from This Tutorial?

By going through this tutorial, you'll learn how to set up a simple ClickHouse cluster. It'll be small, but fault-tolerant and scalable. Then we will use one of the example datasets to fill it with data and execute some demo queries.

## Single Node Setup

To postpone the complexities of a distributed environment, we'll start with deploying ClickHouse on a single server or virtual machine. ClickHouse is usually installed from [deb](#) or [rpm](#) packages, but there are [alternatives](#) for the operating systems that do not support them.

For example, you have chosen deb packages and executed:

```
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.com/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
```

What do we have in the packages that got installed:

- `clickhouse-client` package contains `clickhouse-client` application, interactive ClickHouse console client.
- `clickhouse-common` package contains a ClickHouse executable file.
- `clickhouse-server` package contains configuration files to run ClickHouse as a server.

Server config files are located in `/etc/clickhouse-server/`. Before going further, please notice the `<path>` element in `config.xml`. Path determines the location for data storage, so it should be located on volume with large disk capacity; the default value is `/var/lib/clickhouse/`. If you want to adjust the configuration, it's not handy to directly edit `config.xml` file, considering it might get rewritten on future package updates. The recommended way to override the config elements is to create `files in config.d directory` which serve as "patches" to `config.xml`.

As you might have noticed, `clickhouse-server` is not launched automatically after package installation. It won't be automatically restarted after updates, either. The way you start the server depends on your init system, usually, it is:

```
sudo service clickhouse-server start
```

or

```
sudo /etc/init.d/clickhouse-server start
```

The default location for server logs is `/var/log/clickhouse-server/`. The server is ready to handle client connections once it logs the Ready for connections message.

Once the `clickhouse-server` is up and running, we can use `clickhouse-client` to connect to the server and run some test queries like `SELECT "Hello, world!"`;

#### ► Quick tips for `clickhouse-client`

## Import Sample Dataset

Now it's time to fill our ClickHouse server with some sample data. In this tutorial, we'll use the anonymized data of Yandex.Metrica, the first service that runs ClickHouse in production way before it became open-source (more on that in [history section](#)). There are [multiple ways to import Yandex.Metrica dataset](#), and for the sake of the tutorial, we'll go with the most realistic one.

## Download and Extract Table Data

```
curl https://datasets.clickhouse.com/hits/tsv/hits_v1.tsv.xz | unxz --threads=`nproc` > hits_v1.tsv
curl https://datasets.clickhouse.com/visits/tsv/visits_v1.tsv.xz | unxz --threads=`nproc` > visits_v1.tsv
```

The extracted files are about 10GB in size.

## Create Tables

As in most databases management systems, ClickHouse logically groups tables into "databases". There's a default database, but we'll create a new one named `tutorial`:

```
clickhouse-client --query "CREATE DATABASE IF NOT EXISTS tutorial"
```

Syntax for creating tables is way more complicated compared to databases (see [reference](#)). In general `CREATE TABLE` statement has to specify three key things:

1. Name of table to create.
2. Table schema, i.e. list of columns and their [data types](#).
3. [Table engine](#) and its settings, which determines all the details on how queries to this table will be physically executed.

Yandex.Metrica is a web analytics service, and sample dataset does not cover its full functionality, so there are only two tables to create:

- `hits` is a table with each action done by all users on all websites covered by the service.
- `visits` is a table that contains pre-built sessions instead of individual actions.

Let's see and execute the real create table queries for these tables:

```
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,
    `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,
    Kev5 Strina.
```

```

    `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)

```

```

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,
    `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,
        SearchEngineID UInt16,
        AdvEngineID UInt8,
        PlaceID UInt16,
        SocialSourceNetworkID UInt8,
        Domain String,
        SearchPhrase String).
```

```

        SocialSourceString,
        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)

```

You can execute those queries using the interactive mode of `clickhouse-client` (just launch it in a terminal without specifying a query in advance) or try some [alternative interface](#) if you want.

As we can see, `hits_v1` uses the [basic MergeTree engine](#), while the `visits_v1` uses the [Collapsing](#) variant.

## Import Data

Data import to ClickHouse is done via [INSERT INTO](#) query like in many other SQL databases. However, data is usually provided in one of the [supported serialization formats](#) instead of `VALUES` clause (which is also supported).

The files we downloaded earlier are in tab-separated format, so here's how to import them via console client:

```

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 has a lot of [settings to tune](#) and one way to specify them in console client is via arguments, as we can see with `--max_insert_block_size`. The easiest way to figure out what settings are available, what do they mean and what the defaults are is to query the `system.settings` table:

```
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."
```

Optionally you can [OPTIMIZE](#) the tables after import. Tables that are configured with an engine from MergeTree-family always do merges of data parts in the background to optimize data storage (or at least check if it makes sense). These queries force the table engine to do storage optimization right now instead of some time later:

```
clickhouse-client --query "OPTIMIZE TABLE tutorial.hits_v1 FINAL"
clickhouse-client --query "OPTIMIZE TABLE tutorial.visits_v1 FINAL"
```

These queries start an I/O and CPU intensive operation, so if the table consistently receives new data, it's better to leave it alone and let merges run in the background.

Now we can check if the table import was successful:

```
clickhouse-client --query "SELECT COUNT(*) FROM tutorial.hits_v1"
clickhouse-client --query "SELECT COUNT(*) FROM tutorial.visits_v1"
```

## Example Queries

```
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')
```

## Cluster Deployment

ClickHouse cluster is a homogenous cluster. Steps to set up:

1. Install ClickHouse server on all machines of the cluster
2. Set up cluster configs in configuration files
3. Create local tables on each instance
4. Create a [Distributed table](#)

[Distributed table](#) is actually a kind of “view” to local tables of ClickHouse cluster. SELECT query from a distributed table executes using resources of all cluster’s shards. You may specify configs for multiple clusters and create multiple distributed tables providing views to different clusters.

Example config for a cluster with three shards, one replica each:

```
<remote_servers>
  <perftest_3shards_1replicas>
    <shard>
      <replica>
        <host>example-perftest01j.yandex.ru</host>
        <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>
```

For further demonstration, let's create a new local table with the same `CREATE TABLE` query that we used for `hits_v1`, but different table name:

```
CREATE TABLE tutorial.hits_local (...) ENGINE = MergeTree() ...
```

Creating a distributed table providing a view into local tables of the cluster:

```
CREATE TABLE tutorial.hits_all AS tutorial.hits_local
ENGINE = Distributed(perftest_3shards_1replicas, tutorial, hits_local, rand());
```

A common practice is to create similar Distributed tables on all machines of the cluster. It allows running distributed queries on any machine of the cluster. Also there's an alternative option to create temporary distributed table for a given `SELECT` query using `remote` table function.

Let's run `INSERT SELECT` into the Distributed table to spread the table to multiple servers.

```
INSERT INTO tutorial.hits_all SELECT * FROM tutorial.hits_v1;
```

## Notice

This approach is not suitable for the sharding of large tables. There's a separate tool **clickhouse-copier** that can re-shard arbitrary large tables.

As you could expect, computationally heavy queries run N times faster if they utilize 3 servers instead of one.

In this case, we have used a cluster with 3 shards, and each contains a single replica.

To provide resilience in a production environment, we recommend that each shard should contain 2-3 replicas spread between multiple availability zones or datacenters (or at least racks). Note that ClickHouse supports an unlimited number of replicas.

Example config for a cluster of one shard containing three replicas:

```

<remote_servers>
  ...
  <perftest_1shards_3replicas>
    <shard>
      <replica>
        <host>example-perftest01j.yandex.ru</host>
        <port>9000</port>
      </replica>
      <replica>
        <host>example-perftest02j.yandex.ru</host>
        <port>9000</port>
      </replica>
      <replica>
        <host>example-perftest03j.yandex.ru</host>
        <port>9000</port>
      </replica>
    </shard>
  </perftest_1shards_3replicas>
</remote_servers>

```

To enable native replication **ZooKeeper** is required. ClickHouse takes care of data consistency on all replicas and runs restore procedure after failure automatically. It's recommended to deploy the ZooKeeper cluster on separate servers (where no other processes including ClickHouse are running).

## Note

ZooKeeper is not a strict requirement: in some simple cases, you can duplicate the data by writing it into all the replicas from your application code. This approach is **not** recommended, in this case, ClickHouse won't be able to guarantee data consistency on all replicas. Thus it becomes the responsibility of your application.

ZooKeeper locations are specified in the configuration file:

```

<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>

```

Also, we need to set macros for identifying each shard and replica which are used on table creation:

```

<macros>
  <shard>01</shard>
  <replica>01</replica>
</macros>

```

If there are no replicas at the moment on replicated table creation, a new first replica is instantiated. If there are already live replicas, the new replica clones data from existing ones. You have an option to create all replicated tables first, and then insert data to it. Another option is to create some replicas and add the others after or during data insertion.

```
CREATE TABLE tutorial.hits_replica (...)  
ENGINE = ReplicatedMergeTree(  
    '/clickhouse_perftest/tables/{shard}/hits',  
    '{replica}'  
)  
...
```

Here we use [ReplicatedMergeTree](#) table engine. In parameters we specify ZooKeeper path containing shard and replica identifiers.

```
INSERT INTO tutorial.hits_replica SELECT * FROM tutorial.hits_local;
```

Replication operates in multi-master mode. Data can be loaded into any replica, and the system then syncs it with other instances automatically. Replication is asynchronous so at a given moment, not all replicas may contain recently inserted data. At least one replica should be up to allow data ingestion. Others will sync up data and repair consistency once they will become active again. Note that this approach allows for the low possibility of a loss of recently inserted data.

## ClickHouse Playground

[ClickHouse Playground](#) allows people to experiment with ClickHouse by running queries instantly, without setting up their server or cluster.

Several example datasets are available in Playground as well as sample queries that show ClickHouse features. There's also a selection of ClickHouse LTS releases to experiment with.

ClickHouse Playground gives the experience of m2.small [Managed Service for ClickHouse](#) instance (4 vCPU, 32 GB RAM) hosted in [Yandex.Cloud](#). More information about [cloud providers](#).

You can make queries to Playground using any HTTP client, for example [curl](#) or [wget](#), or set up a connection using [JDBC](#) or [ODBC](#) drivers. More information about software products that support ClickHouse is available [here](#).

## Credentials

Parameter	Value
HTTPS endpoint	<a href="https://play-api.clickhouse.com:8443">https://play-api.clickhouse.com:8443</a>
Native TCP endpoint	play-api.clickhouse.com:9440
User	playground
Password	clickhouse

There are additional endpoints with specific ClickHouse releases to experiment with their differences (ports and user/password are the same as above):

- 20.3 LTS: [play-api-v20-3.clickhouse.com](https://play-api-v20-3.clickhouse.com)
- 19.14 LTS: [play-api-v19-14.clickhouse.com](https://play-api-v19-14.clickhouse.com)

## Note

All these endpoints require a secure TLS connection.

# Limitations

The queries are executed as a read-only user. It implies some limitations:

- DDL queries are not allowed
- INSERT queries are not allowed

The following settings are also enforced:

- `max_result_bytes=10485760`
- `max_result_rows=2000`
- `result_overflow_mode=break`
- `max_execution_time=60000`

## Examples

HTTPS endpoint example with curl:

```
curl "https://play-api.clickhouse.com:8443/?  
query=SELECT+'Play+ClickHouse\!';&user=playground&password=clickhouse&database=datasets"
```

TCP endpoint example with CLI:

```
clickhouse client --secure -h play-api.clickhouse.com --port 9440 -u playground --password clickhouse -q "SELECT 'Play  
ClickHouse\!'"
```

## Implementation Details

ClickHouse Playground web interface makes requests via ClickHouse [HTTP API](#).

The Playground backend is just a ClickHouse cluster without any additional server-side application. As mentioned above, ClickHouse HTTPS and TCP/TLS endpoints are also publicly available as a part of the Playground, both are proxied through [Cloudflare Spectrum](#) to add an extra layer of protection and improved global connectivity.

### Warning

Exposing the ClickHouse server to the public internet in any other situation is **strongly not recommended**. Make sure it listens only on a private network and is covered by a properly configured firewall.

## Interfaces

ClickHouse provides two network interfaces (both can be optionally wrapped in TLS for additional security):

- [HTTP](#), which is documented and easy to use directly.
- [Native TCP](#), which has less overhead.

In most cases it is recommended to use appropriate tool or library instead of interacting with those directly. Officially supported by Yandex are the following:

- [Command-line client](#)
- [JDBC driver](#)
- [ODBC driver](#)
- [C++ client library](#)

There are also a wide range of third-party libraries for working with ClickHouse:

- [Client libraries](#)
- [Integrations](#)
- [Visual interfaces](#)

## Command-line Client

ClickHouse provides a native command-line client: `clickhouse-client`. The client supports command-line options and configuration files. For more information, see [Configuring](#).

[Install](#) it from the `clickhouse-client` package and run it with the command `clickhouse-client`.

```
$ clickhouse-client
ClickHouse client version 20.13.1.5273 (official build).
Connecting to localhost:9000 as user default.
Connected to ClickHouse server version 20.13.1 revision 54442.

:)
```

Different client and server versions are compatible with one another, but some features may not be available in older clients. We recommend using the same version of the client as the server app. When you try to use a client of the older version, then the server, `clickhouse-client` displays the message:

ClickHouse client version is older than ClickHouse server. It may lack support for new features.

## Usage

The client can be used in interactive and non-interactive (batch) mode. To use batch mode, specify the ‘query’ parameter, or send data to ‘stdin’ (it verifies that ‘stdin’ is not a terminal), or both. Similar to the HTTP interface, when using the ‘query’ parameter and sending data to ‘stdin’, the request is a concatenation of the ‘query’ parameter, a line feed, and the data in ‘stdin’. This is convenient for large `INSERT` queries.

Example of using the client to insert data:

```
$ 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";
```

In batch mode, the default data format is TabSeparated. You can set the format in the FORMAT clause of the query.

By default, you can only process a single query in batch mode. To make multiple queries from a “script,” use the `--multiquery` parameter. This works for all queries except INSERT. Query results are output consecutively without additional separators. Similarly, to process a large number of queries, you can run ‘clickhouse-client’ for each query. Note that it may take tens of milliseconds to launch the ‘clickhouse-client’ program.

In interactive mode, you get a command line where you can enter queries.

If ‘multiline’ is not specified (the default): To run the query, press Enter. The semicolon is not necessary at the end of the query. To enter a multiline query, enter a backslash \ before the line feed. After you press Enter, you will be asked to enter the next line of the query.

If multiline is specified: To run a query, end it with a semicolon and press Enter. If the semicolon was omitted at the end of the entered line, you will be asked to enter the next line of the query.

Only a single query is run, so everything after the semicolon is ignored.

You can specify `\G` instead of or after the semicolon. This indicates Vertical format. In this format, each value is printed on a separate line, which is convenient for wide tables. This unusual feature was added for compatibility with the MySQL CLI.

The command line is based on ‘replxx’ (similar to ‘readline’). In other words, it uses the familiar keyboard shortcuts and keeps a history. The history is written to `~/.clickhouse-client-history`.

By default, the format used is PrettyCompact. You can change the format in the FORMAT clause of the query, or by specifying `\G` at the end of the query, using the `--format` or `--vertical` argument in the command line, or using the client configuration file.

To exit the client, press Ctrl+D, or enter one of the following instead of a query: “exit”, “quit”, “logout”, “exit;”, “quit;”, “logout;”, “q”, “Q”, “:q”

When processing a query, the client shows:

1. Progress, which is updated no more than 10 times per second (by default). For quick queries, the progress might not have time to be displayed.
2. The formatted query after parsing, for debugging.
3. The result in the specified format.
4. The number of lines in the result, the time passed, and the average speed of query processing.

You can cancel a long query by pressing Ctrl+C. However, you will still need to wait for a little for the server to abort the request. It is not possible to cancel a query at certain stages. If you do not wait and press Ctrl+C a second time, the client will exit.

The command-line client allows passing external data (external temporary tables) for querying. For more information, see the section “External data for query processing”.

## Queries with Parameters

You can create a query with parameters and pass values to them from client application. This allows to avoid formatting query with specific dynamic values on client side. For example:

```
$ clickhouse-client --param_parName="[1, 2]" -q "SELECT * FROM table WHERE a = {parName:Array(UInt16)}"
```

## Query Syntax

Format a query as usual, then place the values that you want to pass from the app parameters to the query in braces in the following format:

```
{<name>:<data type>}
```

- `name` — Placeholder identifier. In the console client it should be used in app parameters as `--param_<name> = value`.
- `data type` — **Data type** of the app parameter value. For example, a data structure like `(integer, ('string', integer))` can have the `Tuple(UInt8, Tuple(String, UInt8))` data type (you can also use another `integer` types). It's also possible to pass table, database, column names as a parameter, in that case you would need to use `Identifier` as a data type.

## Example

```
$ clickhouse-client --param_tuple_in_tuple="(10, ('dt', 10))" -q "SELECT * FROM table WHERE val = {tuple_in_tuple:Tuple(UInt8, Tuple(String, UInt8))}"  
$ clickhouse-client --param_tbl="numbers" --param_db="system" --param_col="number" --query "SELECT {col:Identifier} FROM {db:Identifier}.{tbl:Identifier} LIMIT 10"
```

# Configuring

You can pass parameters to `clickhouse-client` (all parameters have a default value) using:

- From the Command Line

Command-line options override the default values and settings in configuration files.

- Configuration files.

Settings in the configuration files override the default values.

## Command Line Options

- `--host, -h` – The server name, ‘localhost’ by default. You can use either the name or the IPv4 or IPv6 address.
- `--port` – The port to connect to. Default value: 9000. Note that the HTTP interface and the native interface use different ports.
- `--user, -u` – The username. Default value: default.
- `--password` – The password. Default value: empty string.
- `--query, -q` – The query to process when using non-interactive mode. You must specify either `query` or `queries-file` option.
- `--queries-file, -qf` – file path with queries to execute. You must specify either `query` or `queries-file` option.
- `--database, -d` – Select the current default database. Default value: the current database from the server settings (‘default’ by default).
- `--multiline, -m` – If specified, allow multiline queries (do not send the query on Enter).
- `--multiquery, -n` – If specified, allow processing multiple queries separated by semicolons.
- `--format, -f` – Use the specified default format to output the result.

- `--vertical`, `-E` – If specified, use the **Vertical format** by default to output the result. This is the same as `format=Vertical`. In this format, each value is printed on a separate line, which is helpful when displaying wide tables.
- `--time`, `-t` – If specified, print the query execution time to ‘`stderr`’ in non-interactive mode.
- `--stacktrace` – If specified, also print the stack trace if an exception occurs.
- `--config-file` – The name of the configuration file.
- `--secure` – If specified, will connect to server over secure connection.
- `--history_file` — Path to a file containing command history.
- `--param_<name>` — Value for a **query with parameters**.
- `--hardware-utilization` — Print hardware utilization information in progress bar.
- `--print-profile-events` – Print `ProfileEvents` packets.
- `--profile-events-delay-ms` – Delay between printing `ProfileEvents` packets (-1 - print only totals, 0 - print every single packet).

Since version 20.5, `clickhouse-client` has automatic syntax highlighting (always enabled).

## Configuration Files

`clickhouse-client` uses the first existing file of the following:

- Defined in the `--config-file` parameter.
- `./clickhouse-client.xml`
- `~/.clickhouse-client/config.xml`
- `/etc/clickhouse-client/config.xml`

Example of a config file:

```
<config>
  <user>username</user>
  <password>password</password>
  <secure>False</secure>
</config>
```

## Query ID Format

In interactive mode `clickhouse-client` shows query ID for every query. By default, the ID is formatted like this:

```
Query id: 927f137d-00f1-4175-8914-0dd066365e96
```

A custom format may be specified in a configuration file inside a `query_id_formats` tag. `{query_id}` placeholder in the format string is replaced with the ID of a query. Several format strings are allowed inside the tag. This feature can be used to generate URLs to facilitate profiling of queries.

### Example

```
<config>
<query_id_formats>
  <speedscope>http://speedscope-host/#profileURL=qp%3Fid%3D{query_id}</speedscope>
</query_id_formats>
</config>
```

If the configuration above is applied, the ID of a query is shown in the following format:

```
speedscope:http://speedscope-host/#profileURL=qp%3Fid%3Dc8ecc783-e753-4b38-97f1-42cddfb98b7d
```

## Native Interface (TCP)

The native protocol is used in the [command-line client](#), for inter-server communication during distributed query processing, and also in other C++ programs. Unfortunately, native ClickHouse protocol does not have formal specification yet, but it can be reverse-engineered from ClickHouse source code (starting [around here](#)) and/or by intercepting and analyzing TCP traffic.

## HTTP Interface

The HTTP interface lets you use ClickHouse on any platform from any programming language. We use it for working from Java and Perl, as well as shell scripts. In other departments, the HTTP interface is used from Perl, Python, and Go. The HTTP interface is more limited than the native interface, but it has better compatibility.

By default, `clickhouse-server` listens for HTTP on port 8123 (this can be changed in the config).

If you make a GET / request without parameters, it returns 200 response code and the string which defined in [http\\_server\\_default\\_response](#) default value “Ok.” (with a line feed at the end)

```
$ curl 'http://localhost:8123/'
Ok.
```

Web UI can be accessed here: <http://localhost:8123/play>.

In health-check scripts use GET /ping request. This handler always returns “Ok.” (with a line feed at the end). Available from version 18.12.13.

```
$ curl 'http://localhost:8123/ping'
Ok.
```

Send the request as a URL ‘query’ parameter, or as a POST. Or send the beginning of the query in the ‘query’ parameter, and the rest in the POST (we’ll explain later why this is necessary). The size of the URL is limited to 16 KB, so keep this in mind when sending large queries.

If successful, you receive the 200 response code and the result in the response body.

If an error occurs, you receive the 500 response code and an error description text in the response body.

When using the GET method, ‘readonly’ is set. In other words, for queries that modify data, you can only use the POST method. You can send the query itself either in the POST body or in the URL parameter.

Examples:

```
$ curl 'http://localhost:8123/?query=SELECT%201'
1

$ wget -nv -O- '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
Date: Wed, 27 Nov 2019 10:30:18 GMT
Connection: Close
Content-Type: text/tab-separated-values; charset=UTF-8
X-ClickHouse-Server-Display-Name: clickhouse.ru-central1.internal
X-ClickHouse-Query-Id: 5abe861c-239c-467f-b955-8a201abb8b7f
X-ClickHouse-Summary:
{"read_rows":"0","read_bytes":"0","written_rows":"0","written_bytes":"0","total_rows_to_read":"0"}

1
```

As you can see, `curl` is somewhat inconvenient in that spaces must be URL escaped.

Although `wget` escapes everything itself, we do not recommend using it because it does not work well over HTTP 1.1 when using keep-alive and Transfer-Encoding: chunked.

```
$ 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
```

If part of the query is sent in the parameter, and part in the POST, a line feed is inserted between these two data parts.

Example (this won't work):

```
$ 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
```

By default, data is returned in **TabSeparated** format.

You use the **FORMAT** clause of the query to request any other format.

Also, you can use the '`default_format`' URL parameter or the '`X-ClickHouse-Format`' header to specify a default format other than TabSeparated.

```
$ echo 'SELECT 1 FORMAT Pretty' | curl 'http://localhost:8123/?' --data-binary @-
```

The POST method of transmitting data is necessary for `INSERT` queries. In this case, you can write the beginning of the query in the URL parameter, and use POST to pass the data to insert. The data to insert could be, for example, a tab-separated dump from MySQL. In this way, the `INSERT` query replaces `LOAD DATA LOCAL INFILE` from MySQL.

## Examples

Creating a table:

```
$ echo 'CREATE TABLE t (a UInt8) ENGINE = Memory' | curl 'http://localhost:8123/' --data-binary @-
```

Using the familiar INSERT query for data insertion:

```
$ echo 'INSERT INTO t VALUES (1),(2),(3)' | curl 'http://localhost:8123/' --data-binary @-
```

Data can be sent separately from the query:

```
$ echo '(4),(5),(6)' | curl 'http://localhost:8123/?query=INSERT%20INTO%20t%20VALUES' --data-binary @-
```

You can specify any data format. The ‘Values’ format is the same as what is used when writing INSERT INTO t VALUES:

```
$ echo '(7),(8),(9)' | curl 'http://localhost:8123/?query=INSERT%20INTO%20t%20FORMAT%20Values' --data-binary @-
```

To insert data from a tab-separated dump, specify the corresponding format:

```
$ echo -ne '10\n11\n12\n' | curl 'http://localhost:8123/?query=INSERT%20INTO%20t%20FORMAT%20TabSeparated' --data-binary @-
```

Reading the table contents. Data is output in random order due to parallel query processing:

```
$ curl 'http://localhost:8123/?query=SELECT%20a%20FROM%20t'  
7  
8  
9  
10  
11  
12  
1  
2  
3  
4  
5  
6
```

Deleting the table.

```
$ echo 'DROP TABLE t' | curl 'http://localhost:8123/' --data-binary @-
```

For successful requests that do not return a data table, an empty response body is returned.

## Compression

You can use compression to reduce network traffic when transmitting a large amount of data or for creating dumps that are immediately compressed.

You can use the internal ClickHouse compression format when transmitting data. The compressed data has a non-standard format, and you need `clickhouse-compressor` program to work with it. It is installed with the `clickhouse-client` package. To increase the efficiency of data insertion, you can disable server-side checksum verification by using the `http_native_compression_disable_checksumming_on_decompress` setting.

If you specify `compress=1` in the URL, the server will compress the data it sends to you. If you specify `decompress=1` in the URL, the server will decompress the data which you pass in the `POST` method.

You can also choose to use [HTTP compression](#). ClickHouse supports the following [compression methods](#):

- gzip
- br
- deflate
- xz

To send a compressed POST request, append the request header `Content-Encoding: compression_method`. In order for ClickHouse to compress the response, enable compression with `enable_http_compression` setting and append `Accept-Encoding: compression_method` header to the request. You can configure the data compression level in the `http_zlib_compression_level` setting for all compression methods.

## Note

Some HTTP clients might decompress data from the server by default (with `gzip` and `deflate`) and you might get decompressed data even if you use the compression settings correctly.

## Examples

```
## Sending compressed data to the server
$ echo "SELECT 1" | gzip -c | \
curl -sS --data-binary @- -H 'Content-Encoding: gzip' 'http://localhost:8123/'
```

```
## Receiving compressed data from the server
$ curl -vsS "http://localhost:8123/?enable_http_compression=1" \
-H 'Accept-Encoding: gzip' --output result.gz -d 'SELECT number FROM system.numbers LIMIT 3'
$ zcat result.gz
0
1
2
```

## Default Database

You can use the ‘database’ URL parameter or the ‘X-ClickHouse-Database’ header to specify the default database.

```
$ 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
```

By default, the database that is registered in the server settings is used as the default database. By default, this is the database called ‘default’. Alternatively, you can always specify the database using a dot before the table name.

The username and password can be indicated in one of three ways:

1. Using HTTP Basic Authentication. Example:

```
$ echo 'SELECT 1' | curl 'http://user:password@localhost:8123/' -d @-
```

1. In the ‘user’ and ‘password’ URL parameters. Example:

```
$ echo 'SELECT 1' | curl 'http://localhost:8123/?user=user&password=password' -d @-
```

1. Using ‘X-ClickHouse-User’ and ‘X-ClickHouse-Key’ headers. Example:

```
$ echo 'SELECT 1' | curl -H 'X-ClickHouse-User: user' -H 'X-ClickHouse-Key: password' 'http://localhost:8123/' -d @-
```

If the user name is not specified, the `default` name is used. If the password is not specified, the empty password is used.

You can also use the URL parameters to specify any settings for processing a single query or entire profiles of settings. Example:<http://localhost:8123/>

`profile=web&max_rows_to_read=1000000000&query=SELECT+1`

For more information, see the [Settings](#) section.

```
$ echo 'SELECT number FROM system.numbers LIMIT 10' | curl 'http://localhost:8123/?' --data-binary @-
0
1
2
3
4
5
6
7
8
9
```

For information about other parameters, see the section “[SET](#)”.

Similarly, you can use ClickHouse sessions in the HTTP protocol. To do this, you need to add the `session_id` GET parameter to the request. You can use any string as the session ID. By default, the session is terminated after 60 seconds of inactivity. To change this timeout, modify the `default_session_timeout` setting in the server configuration, or add the `session_timeout` GET parameter to the request. To check the session status, use the `session_check=1` parameter. Only one query at a time can be executed within a single session.

You can receive information about the progress of a query in `X-ClickHouse-Progress` response headers. To do this, enable [send\\_progress\\_in\\_http\\_headers](#). Example of the header sequence:

```
X-ClickHouse-Progress: {"read_rows": "2752512", "read_bytes": "240570816", "total_rows_to_read": "8880128"}
X-ClickHouse-Progress: {"read_rows": "5439488", "read_bytes": "482285394", "total_rows_to_read": "8880128"}
X-ClickHouse-Progress: {"read_rows": "8783786", "read_bytes": "819092887", "total_rows_to_read": "8880128"}
```

Possible header fields:

- `read_rows` — Number of rows read.
- `read_bytes` — Volume of data read in bytes.
- `total_rows_to_read` — Total number of rows to be read.
- `written_rows` — Number of rows written.
- `written_bytes` — Volume of data written in bytes.

Running requests do not stop automatically if the HTTP connection is lost. Parsing and data formatting are performed on the server-side, and using the network might be ineffective.

The optional ‘query\_id’ parameter can be passed as the query ID (any string). For more information, see the section “Settings, replace\_running\_query”.

The optional ‘quota\_key’ parameter can be passed as the quota key (any string). For more information, see the section “Quotas”.

The HTTP interface allows passing external data (external temporary tables) for querying. For more information, see the section “External data for query processing”.

## Response Buffering

You can enable response buffering on the server-side. The `buffer_size` and `wait_end_of_query` URL parameters are provided for this purpose.

`buffer_size` determines the number of bytes in the result to buffer in the server memory. If a result body is larger than this threshold, the buffer is written to the HTTP channel, and the remaining data is sent directly to the HTTP channel.

To ensure that the entire response is buffered, set `wait_end_of_query=1`. In this case, the data that is not stored in memory will be buffered in a temporary server file.

Example:

```
$ 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'
```

Use buffering to avoid situations where a query processing error occurred after the response code and HTTP headers were sent to the client. In this situation, an error message is written at the end of the response body, and on the client-side, the error can only be detected at the parsing stage.

## Queries with Parameters

You can create a query with parameters and pass values for them from the corresponding HTTP request parameters. For more information, see [Queries with Parameters for CLI](#).

Example

```
$ curl -sS "<address>?param_id=2&param_phrase=test" -d "SELECT * FROM table WHERE int_column = {id:UInt8}  
and string_column = {phrase:String}"
```

## Predefined HTTP Interface

ClickHouse supports specific queries through the HTTP interface. For example, you can write data to a table as follows:

```
$ echo '(4),(5),(6)' | curl 'http://localhost:8123/?query=INSERT%20INTO%20t%20VALUES' --data-binary @-
```

ClickHouse also supports Predefined HTTP Interface which can help you more easily integrate with third-party tools like [Prometheus exporter](#).

Example:

- First of all, add this section to server configuration file:

```

<http_handlers>
  <rule>
    <url>/predefined_query</url>
    <methods>POST,GET</methods>
    <handler>
      <type>predefined_query_handler</type>
      <query>SELECT * FROM system.metrics LIMIT 5 FORMAT Template SETTINGS format_template_resultset = 'prometheus_template_output_format_resultset', format_template_row = 'prometheus_template_output_format_row', format_template_rows_between_delimiter = '\n'</query>
    </handler>
  </rule>
  <rule>...</rule>
  <rule>...</rule>
</http_handlers>

```

- You can now request the URL directly for data in the Prometheus format:

```

$ curl -v 'http://localhost:8123/predefined_query'
* Trying ::1...
* Connected to localhost (::1) port 8123 (#0)
> GET /predefined_query HTTP/1.1
> Host: localhost:8123
> User-Agent: curl/7.47.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Tue, 28 Apr 2020 08:52:56 GMT
< Connection: Keep-Alive
< Content-Type: text/plain; charset=UTF-8
< X-ClickHouse-Server-Display-Name: i-mloy5trc
< Transfer-Encoding: chunked
< X-ClickHouse-Query-Id: 96fe0052-01e6-43ce-b12a-6b7370de6e8a
< X-ClickHouse-Format: Template
< X-ClickHouse-Timezone: Asia/Shanghai
< Keep-Alive: timeout=3
< X-ClickHouse-Summary:
{"read_rows":"0","read_bytes":"0","written_rows":"0","written_bytes":"0","total_rows_to_read":"0"}
<
## HELP "Query" "Number of executing queries"
## TYPE "Query" counter
"Query" 1

## HELP "Merge" "Number of executing background merges"
## TYPE "Merge" counter
"Merge" 0

## HELP "PartMutation" "Number of mutations (ALTER DELETE/UPDATE)"
## TYPE "PartMutation" counter
"PartMutation" 0

## HELP "ReplicatedFetch" "Number of data parts being fetched from replica"
## TYPE "ReplicatedFetch" counter
"ReplicatedFetch" 0

## HELP "ReplicatedSend" "Number of data parts being sent to replicas"
## TYPE "ReplicatedSend" counter
"ReplicatedSend" 0

* Connection #0 to host localhost left intact
* Connection #0 to host localhost left intact

```

As you can see from the example if `http_handlers` is configured in the `config.xml` file and `http_handlers` can contain many rules. ClickHouse will match the HTTP requests received to the predefined type in rule and the first matched runs the handler. Then ClickHouse will execute the corresponding predefined query if the match is successful.

Now rule can configure method, headers, url, handler:

- method is responsible for matching the method part of the HTTP request. method fully conforms to the definition of method in the HTTP protocol. It is an optional configuration. If it is not defined in the configuration file, it does not match the method portion of the HTTP request.
- url is responsible for matching the URL part of the HTTP request. It is compatible with RE2's regular expressions. It is an optional configuration. If it is not defined in the configuration file, it does not match the URL portion of the HTTP request.
- headers are responsible for matching the header part of the HTTP request. It is compatible with RE2's regular expressions. It is an optional configuration. If it is not defined in the configuration file, it does not match the header portion of the HTTP request.
- handler contains the main processing part. Now handler can configure type, status, content\_type, response\_content, query, query\_param\_name.  
type currently supports three types: predefined\_query\_handler, dynamic\_query\_handler, static.
  - query — use with predefined\_query\_handler type, executes query when the handler is called.
  - query\_param\_name — use with dynamic\_query\_handler type, extracts and executes the value corresponding to the query\_param\_name value in HTTP request params.
  - status — use with static type, response status code.
  - content\_type — use with static type, response content-type.
  - response\_content — use with static type, response content sent to client, when using the prefix 'file://' or 'config://', find the content from the file or configuration sends to client.

Next are the configuration methods for different type.

## predefined\_query\_handler

predefined\_query\_handler supports setting Settings and query\_params values. You can configure query in the type of predefined\_query\_handler.

query value is a predefined query of predefined\_query\_handler, which is executed by ClickHouse when an HTTP request is matched and the result of the query is returned. It is a must configuration.

The following example defines the values of max\_threads and max\_alter\_threads settings, then queries the system table to check whether these settings were set successfully.

Example:

```
<http_handlers>
  <rule>
    <url><![CDATA[/query_param_with_url/w+/(?P<name_1>[^/]+)(/(?P<name_2>[^/]+))?]]></url>
    <method>GET</method>
    <headers>
      <XXX>TEST_HEADER_VALUE</XXX>
      <PARAMS_XXX><![CDATA[({?P<name_1>[^/]+}(/{?P<name_2>[^/]+})?)]]></PARAMS_XXX>
    </headers>
    <handler>
      <type>predefined_query_handler</type>
      <query>SELECT value FROM system.settings WHERE name = {name_1:String} </query>
      <query>SELECT name, value FROM system.settings WHERE name = {name_2:String}</query>
    </handler>
  </rule>
</http_handlers>
```

```
$ curl -H 'XXX:TEST_HEADER_VALUE' -H 'PARAMS_XXX:max_threads'
'http://localhost:8123/query_param_with_url/1/max_threads/max_alter_threads?
max_threads=1&max_alter_threads=2'
1
max_alter_threads 2
```

## caution

In one `predefined_query_handler` only supports one query of an insert type.

## dynamic\_query\_handler

In `dynamic_query_handler`, the query is written in the form of param of the HTTP request. The difference is that in `predefined_query_handler`, the query is written in the configuration file. You can configure `query_param_name` in `dynamic_query_handler`.

ClickHouse extracts and executes the value corresponding to the `query_param_name` value in the URL of the HTTP request. The default value of `query_param_name` is `/query`. It is an optional configuration. If there is no definition in the configuration file, the param is not passed in.

To experiment with this functionality, the example defines the values of `max_threads` and `max_alter_threads` and queries whether the settings were set successfully.

Example:

```
<http_handlers>
  <rule>
    <headers>
      <XXX>TEST_HEADER_VALUE_DYNAMIC</XXX>  </headers>
    <handler>
      <type>dynamic_query_handler</type>
      <query_param_name>query_param</query_param_name>
    </handler>
  </rule>
</http_handlers>
```

```
$ curl -H 'XXX:TEST_HEADER_VALUE_DYNAMIC' 'http://localhost:8123/own?
max_threads=1&max_alter_threads=2&param_name_1=max_threads&param_name_2=max_alter_threads&query_p
aram=SELECT%20name,value%20FROM%20system.settings%20where%20name%20=%20%7Bname_1:String%7D%
20OR%20name%20=%20%7Bname_2:String%7D'
max_threads 1
max_alter_threads 2
```

## static

`static` can return `content_type`, `status` and `response_content`. `response_content` can return the specified content.

Example:

Return a message.

```

<http_handlers>
  <rule>
    <methods>GET</methods>
    <headers><XXX>xxx</XXX></headers>
    <url>/hi</url>
    <handler>
      <type>static</type>
      <status>402</status>
      <content_type>text/html; charset=UTF-8</content_type>
      <response_content>Say Hi!</response_content>
    </handler>
  </rule>
</http_handlers>

```

```

$ curl -vv -H 'XXX:xxx' 'http://localhost:8123/hi'
* Trying ::1...
* Connected to localhost (::1) port 8123 (#0)
> GET /hi HTTP/1.1
> Host: localhost:8123
> User-Agent: curl/7.47.0
> Accept: /*
> XXX:xxx
>
< HTTP/1.1 402 Payment Required
< Date: Wed, 29 Apr 2020 03:51:26 GMT
< Connection: Keep-Alive
< Content-Type: text/html; charset=UTF-8
< Transfer-Encoding: chunked
< Keep-Alive: timeout=3
< X-ClickHouse-Summary:
{ "read_rows": "0", "read_bytes": "0", "written_rows": "0", "written_bytes": "0", "total_rows_to_read": "0" }
<
* Connection #0 to host localhost left intact
Say Hi!%

```

Find the content from the configuration send to client.

```

<get_config_static_handler><![CDATA[<html ng-app="SMI2"><head><base href="http://ui.tabix.io/"></head>
<body><div ui-view="" class="content-ui"></div><script src="http://loader.tabix.io/master.js"></script></body>
</html>]]></get_config_static_handler>

<http_handlers>
  <rule>
    <methods>GET</methods>
    <headers><XXX>xxx</XXX></headers>
    <url>/get_config_static_handler</url>
    <handler>
      <type>static</type>
      <response_content>config://get_config_static_handler</response_content>
    </handler>
  </rule>
</http_handlers>

```

```

$ curl -v -H 'XXX:xxx' 'http://localhost:8123/get_config_static_handler'
* Trying ::1...
* Connected to localhost (::1) port 8123 (#0)
> GET /get_config_static_handler HTTP/1.1
> Host: localhost:8123
> User-Agent: curl/7.47.0
> Accept: /*
> XXX:xxx
>
< HTTP/1.1 200 OK
< Date: Wed, 29 Apr 2020 04:01:24 GMT
< Connection: Keep-Alive
< Content-Type: text/plain; charset=UTF-8
< Transfer-Encoding: chunked
< Keep-Alive: timeout=3
< X-ClickHouse-Summary:
{"read_rows":"0","read_bytes":"0","written_rows":"0","written_bytes":"0","total_rows_to_read":"0"}
<
* Connection #0 to host localhost left intact
<html ng-app="SMI2"><head><base href="http://ui.tabix.io/"></head><body><div ui-view="" class="content-ui"></div><script src="http://loader.tabix.io/master.js"></script></body></html>%

```

Find the content from the file send to client.

```

<http_handlers>
  <rule>
    <methods>GET</methods>
    <headers><XXX>xxx</XXX></headers>
    <url>/get_absolute_path_static_handler</url>
    <handler>
      <type>static</type>
      <content_type>text/html; charset=UTF-8</content_type>
      <response_content>file:///absolute_path_file.html</response_content>
    </handler>
  </rule>
  <rule>
    <methods>GET</methods>
    <headers><XXX>xxx</XXX></headers>
    <url>/get_relative_path_static_handler</url>
    <handler>
      <type>static</type>
      <content_type>text/html; charset=UTF-8</content_type>
      <response_content>file://./relative_path_file.html</response_content>
    </handler>
  </rule>
</http_handlers>

```

```
$ user_files_path='/var/lib/clickhouse/user_files'
$ sudo echo "<html><body>Relative Path File</body></html>" > $user_files_path/relative_path_file.html
$ sudo echo "<html><body>Absolute Path File</body></html>" > $user_files_path/absolute_path_file.html
$ curl -vv -H 'XXX:xxx' 'http://localhost:8123/get_absolute_path_static_handler'
* Trying ::1...
* Connected to localhost (::1) port 8123 (#0)
> GET /get_absolute_path_static_handler HTTP/1.1
> Host: localhost:8123
> User-Agent: curl/7.47.0
> Accept: /*
> XXX:xxx
>
< HTTP/1.1 200 OK
< Date: Wed, 29 Apr 2020 04:18:16 GMT
< Connection: Keep-Alive
< Content-Type: text/html; charset=UTF-8
< Transfer-Encoding: chunked
< Keep-Alive: timeout=3
< X-ClickHouse-Summary:
>{"read_rows":"0","read_bytes":"0","written_rows":"0","written_bytes":"0","total_rows_to_read":"0"}
<
<html><body>Absolute Path File</body></html>
* Connection #0 to host localhost left intact
$ curl -vv -H 'XXX:xxx' 'http://localhost:8123/get_relative_path_static_handler'
* Trying ::1...
* Connected to localhost (::1) port 8123 (#0)
> GET /get_relative_path_static_handler HTTP/1.1
> Host: localhost:8123
> User-Agent: curl/7.47.0
> Accept: /*
> XXX:xxx
>
< HTTP/1.1 200 OK
< Date: Wed, 29 Apr 2020 04:18:31 GMT
< Connection: Keep-Alive
< Content-Type: text/html; charset=UTF-8
< Transfer-Encoding: chunked
< Keep-Alive: timeout=3
< X-ClickHouse-Summary:
>{"read_rows":"0","read_bytes":"0","written_rows":"0","written_bytes":"0","total_rows_to_read":"0"}
<
<html><body>Relative Path File</body></html>
* Connection #0 to host localhost left intact
```

## MySQL Interface

ClickHouse supports MySQL wire protocol. It can be enabled by `mysql_port` setting in configuration file:

```
<mysql_port>9004</mysql_port>
```

Example of connecting using command-line tool `mysql`:

```
$ mysql --protocol tcp -u default -P 9004
```

Output if a connection succeeded:

```
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>
```

For compatibility with all MySQL clients, it is recommended to specify user password with **double SHA1** in configuration file.

If user password is specified using **SHA256**, some clients won't be able to authenticate (mysqljs and old versions of command-line tool mysql).

Restrictions:

- prepared queries are not supported
- some data types are sent as strings

To cancel a long query use `KILL QUERY connection_id` statement (it is replaced with `KILL QUERY WHERE query_id = connection_id` while proceeding). For example:

```
$ mysql --protocol tcp -h mysql_server -P 9004 default -u default --password=123 -e "KILL QUERY 123456;"
```

## Formats for Input and Output Data

ClickHouse can accept and return data in various formats. A format supported for input can be used to parse the data provided to `INSERTs`, to perform `SELECTs` from a file-backed table such as File, URL or HDFS, or to read an external dictionary. A format supported for output can be used to arrange the results of a `SELECT`, and to perform `INSERTs` into a file-backed table.

The supported formats are:

Format	Input	Output
TabSeparated	✓	✓
TabSeparatedRaw	✓	✓
TabSeparatedWithNames	✓	✓
TabSeparatedWithNamesAndTypes	✓	✓
Template	✓	✓
TemplateIgnoreSpaces	✓	✗
CSV	✓	✓
CSVWithNames	✓	✓

Format	Input	Output
CustomSeparated	✓	✓
Values	✓	✓
Vertical	✗	✓
JSON	✗	✓
JSONAsString	✓	✗
JSONStrings	✗	✓
JSONCompact	✗	✓
JSONCompactStrings	✗	✓
JSONEachRow	✓	✓
JSONEachRowWithProgress	✗	✓
JSONStringsEachRow	✓	✓
JSONStringsEachRowWithProgress	✗	✓
JSONCompactEachRow	✓	✓
JSONCompactEachRowWithNamesAndTypes	✓	✓
JSONCompactStringsEachRow	✓	✓
JSONCompactStringsEachRowWithNamesAndTypes	✓	✓
TSKV	✓	✓
Pretty	✗	✓
PrettyCompact	✗	✓
PrettyCompactMonoBlock	✗	✓
PrettyNoEscapes	✗	✓
PrettySpace	✗	✓
Protobuf	✓	✓
ProtobufSingle	✓	✓
Avro	✓	✓

Format	Input	Output
AvroConfluent	✓	✗
Parquet	✓	✓
Arrow	✓	✓
ArrowStream	✓	✓
ORC	✓	✓
RowBinary	✓	✓
RowBinaryWithNamesAndTypes	✓	✓
Native	✓	✓
Null	✗	✓
XML	✗	✓
CapnProto	✓	✗
LineAsString	✓	✗
Regexp	✓	✗
RawBLOB	✓	✓
MsgPack	✓	✓

You can control some format processing parameters with the ClickHouse settings. For more information read the [Settings](#) section.

## TabSeparated

In TabSeparated format, data is written by row. Each row contains values separated by tabs. Each value is followed by a tab, except the last value in the row, which is followed by a line feed. Strictly Unix line feeds are assumed everywhere. The last row also must contain a line feed at the end. Values are written in text format, without enclosing quotation marks, and with special characters escaped.

This format is also available under the name [TSV](#).

The TabSeparated format is convenient for processing data using custom programs and scripts. It is used by default in the HTTP interface, and in the command-line client's batch mode. This format also allows transferring data between different DBMSs. For example, you can get a dump from MySQL and upload it to ClickHouse, or vice versa.

The TabSeparated format supports outputting total values (when using WITH TOTALS) and extreme values (when 'extremes' is set to 1). In these cases, the total values and extremes are output after the main data. The main result, total values, and extremes are separated from each other by an empty line. Example:

```
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
1970-01-01	8873898
2014-03-17	1031592
2014-03-23	1406958

## Data Formatting

Integer numbers are written in decimal form. Numbers can contain an extra “+” character at the beginning (ignored when parsing, and not recorded when formatting). Non-negative numbers can’t contain the negative sign. When reading, it is allowed to parse an empty string as a zero, or (for signed types) a string consisting of just a minus sign as a zero. Numbers that do not fit into the corresponding data type may be parsed as a different number, without an error message.

Floating-point numbers are written in decimal form. The dot is used as the decimal separator. Exponential entries are supported, as are ‘inf’, ‘+inf’, ‘-inf’, and ‘nan’. An entry of floating-point numbers may begin or end with a decimal point.

During formatting, accuracy may be lost on floating-point numbers.

During parsing, it is not strictly required to read the nearest machine-representable number.

Dates are written in YYYY-MM-DD format and parsed in the same format, but with any characters as separators.

Dates with times are written in the format YYYY-MM-DD hh:mm:ss and parsed in the same format, but with any characters as separators.

This all occurs in the system time zone at the time the client or server starts (depending on which of them formats data). For dates with times, daylight saving time is not specified. So if a dump has times during daylight saving time, the dump does not unequivocally match the data, and parsing will select one of the two times.

During a read operation, incorrect dates and dates with times can be parsed with natural overflow or as null dates and times, without an error message.

As an exception, parsing dates with times is also supported in Unix timestamp format, if it consists of exactly 10 decimal digits. The result is not time zone-dependent. The formats YYYY-MM-DD hh:mm:ss and NNNNNNNNNN are differentiated automatically.

Strings are output with backslash-escaped special characters. The following escape sequences are used for output: \b, \f, \r, \n, \t, \0, \', \\. Parsing also supports the sequences \a, \v, and \xHH (hex escape sequences) and any \c sequences, where c is any character (these sequences are converted to c). Thus, reading data supports formats where a line feed can be written as \n or \, or as a line feed. For example, the string Hello world with a line feed between the words instead of space can be parsed in any of the following variations:

```
Hello\nworld
```

```
Hello\  
world
```

The second variant is supported because MySQL uses it when writing tab-separated dumps.

The minimum set of characters that you need to escape when passing data in TabSeparated format: tab, line feed (LF) and backslash.

Only a small set of symbols are escaped. You can easily stumble onto a string value that your terminal will ruin in output.

Arrays are written as a list of comma-separated values in square brackets. Number items in the array are formatted as normally. `Date` and `DateTime` types are written in single quotes. Strings are written in single quotes with the same escaping rules as above.

`NULL` is formatted as `\N`.

Each element of `Nested` structures is represented as array.

For example:

```
CREATE TABLE nestedt
(
    `id` UInt8,
    `aux` Nested(
        a UInt8,
        b String
    )
)
ENGINE = TinyLog
```

```
INSERT INTO nestedt Values ( 1, [1], ['a'])
```

```
SELECT * FROM nestedt FORMAT TSV
```

```
1 [1] ['a']
```

## TabSeparatedRaw

Differs from `TabSeparated` format in that the rows are written without escaping.  
When parsing with this format, tabs or linefeeds are not allowed in each field.

This format is also available under the name `TSVRaw`.

## TabSeparatedWithNames

Differs from the `TabSeparated` format in that the column names are written in the first row.  
During parsing, the first row is expected to contain the column names. You can use column names to determine their position and to check their correctness.

This format is also available under the name `TSVWithNames`.

## TabSeparatedWithNamesAndTypes

Differs from the `TabSeparated` format in that the column names are written to the first row, while the column types are in the second row.

During parsing, the first and second rows are completely ignored.

This format is also available under the name `TSVWithNamesAndTypes`.

## Template

This format allows specifying a custom format string with placeholders for values with a specified escaping rule.

It uses settings `format_template_resultset`, `format_template_row`, `format_template_rows_between_delimiter` and some settings of other formats (e.g. `output_format_json_quote_64bit_integers` when using JSON escaping, see further)

Setting `format_template_row` specifies path to file, which contains format string for rows with the following syntax:

```
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 or index 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 an 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;
```

The `format_template_rows_between_delimiter` setting specifies delimiter between rows, which is printed (or expected) after every row except the last one (\n by default)

Setting `format_template_resultset` specifies the path to file, which contains a format string for resultset. Format string for resultset has the same syntax as a format string for row and allows to specify a prefix, a suffix and a way to print some additional information. It contains the following placeholders instead of column names:

- `data` is the rows with data in `format_template_row` format, separated by `format_template_rows_between_delimiter`. This placeholder must be the first placeholder in the format string.
- `totals` is the row with total values in `format_template_row` format (when using `WITH TOTALS`)
- `min` is the row with minimum values in `format_template_row` format (when extremes are set to 1)
- `max` is the row with maximum values in `format_template_row` format (when extremes are set to 1)
- `rows` is the total number of output rows
- `rows_before_limit` is the minimal number of rows there would have been without LIMIT. Output only if the query contains LIMIT. If the query contains GROUP BY, `rows_before_limit_at_least` is the exact number of rows there would have been without a LIMIT.

- `time` is the request execution time in seconds
- `rows_read` is the number of rows has been read
- `bytes_read` is the number of bytes (uncompressed) has been read

The placeholders `data`, `totals`, `min` and `max` must not have escaping rule specified (or `None` must be specified explicitly). The remaining placeholders may have any escaping rule specified.

If the `format_template_resultset` setting is an empty string,  `${data}`  is used as default value.

For insert queries format allows skipping some columns or some fields if prefix or suffix (see example).

Select example:

```
SELECT SearchPhrase, count() AS c FROM test.hits GROUP BY SearchPhrase ORDER BY c DESC LIMIT 5 FORMAT
Template SETTINGS
format_template_resultset = '/some/path/resultset.format', format_template_row = '/some/path/row.format',
format_template_rows_between_delimiter = '\n '
```

/some/path/resultset.format:

```
<!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>
```

/some/path/row.format:

```
<tr> <td>${0:XML}</td> <td>${1:XML}</td> </tr>
```

Result:

```
<!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 example:

```
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_template_resultset = '/some/path/resultset.format', format_template_row = '/some/path/row.format'
```

```
/some/path/resultset.format:
```

```
Some header\n${data}\nTotal rows: ${:CSV}\n
```

```
/some/path/row.format:
```

```
Page views: ${PageViews:CSV}, User id: ${UserID:CSV}, Useless field: ${:CSV}, Duration: ${Duration:CSV}, Sign: ${Sign:CSV}
```

PageViews, UserID, Duration and Sign inside placeholders are names of columns in the table. Values after Useless field in rows and after \nTotal rows: in suffix will be ignored.

All delimiters in the input data must be strictly equal to delimiters in specified format strings.

## TemplateIgnoreSpaces

This format is suitable only for input.

Similar to [Template](#), but skips whitespace characters between delimiters and values in the input stream. However, if format strings contain whitespace characters, these characters will be expected in the input stream. Also allows to specify empty placeholders \${:} or \${:None} to split some delimiter into separate parts to ignore spaces between them. Such placeholders are used only for skipping whitespace characters. It's possible to read [JSON](#) using this format, if values of columns have the same order in all rows. For example, the following request can be used for inserting data from output example of format [JSON](#):

```
INSERT INTO table_name FORMAT TemplateIgnoreSpaces SETTINGS
format_template_resultset = '/some/path/resultset.format', format_template_row = '/some/path/row.format',
format_template_rows_between_delimiter = ','
```

```
/some/path/resultset.format:
```

```
 ${{}"meta":${}:${:JSON},${}"data":${}:${}
[${data}]${},${}"totals":${}:${:JSON},${}"extremes":${}:${:JSON},${}"rows":${}:${:JSON},${}"rows_before_limit_
at_least":${}:${:JSON}${}}
```

```
/some/path/row.format:
```

```
 ${{}"SearchPhrase":${}:${}{phrase:JSON}${},${}"c":${}:${}{cnt:JSON}${}}
```

## TSKV

Similar to [TabSeparated](#), but outputs a value in name=value format. Names are escaped the same way as in [TabSeparated](#) format, and the = symbol is also escaped.

```
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` is formatted as `\N`.

```
SELECT * FROM t_null FORMAT TSKV
```

```
x=1 y=\N
```

When there is a large number of small columns, this format is ineffective, and there is generally no reason to use it. Nevertheless, it is no worse than `JSONEachRow` in terms of efficiency.

Both data output and parsing are supported in this format. For parsing, any order is supported for the values of different columns. It is acceptable for some values to be omitted – they are treated as equal to their default values. In this case, zeros and blank rows are used as default values. Complex values that could be specified in the table are not supported as defaults.

Parsing allows the presence of the additional field `tskv` without the equal sign or a value. This field is ignored.

## CSV

Comma Separated Values format ([RFC](#)).

When formatting, rows are enclosed in double-quotes. A double quote inside a string is output as two double quotes in a row. There are no other rules for escaping characters. Date and date-time are enclosed in double-quotes. Numbers are output without quotes. Values are separated by a delimiter character, which is `,` by default. The delimiter character is defined in the setting `format_csv_delimiter`. Rows are separated using the Unix line feed (LF). Arrays are serialized in CSV as follows: first, the array is serialized to a string as in TabSeparated format, and then the resulting string is output to CSV in double-quotes. Tuples in CSV format are serialized as separate columns (that is, their nesting in the tuple is lost).

```
$ clickhouse-client --format_csv_delimiter="|" --query="INSERT INTO test.csv FORMAT CSV" < data.csv
```

\*By default, the delimiter is `,`. See the `format_csv_delimiter` setting for more information.

When parsing, all values can be parsed either with or without quotes. Both double and single quotes are supported. Rows can also be arranged without quotes. In this case, they are parsed up to the delimiter character or line feed (CR or LF). In violation of the RFC, when parsing rows without quotes, the leading and trailing spaces and tabs are ignored. For the line feed, Unix (LF), Windows (CR LF) and Mac OS Classic (CR LF) types are all supported.

Empty unquoted input values are replaced with default values for the respective columns, if `input_format_defaults_for_omitted_fields` is enabled.

`NULL` is formatted as `\N` or `NULL` or an empty unquoted string (see settings `input_format_csv_unquoted_null_literal_as_null` and `input_format_defaults_for_omitted_fields`).

The CSV format supports the output of totals and extremes the same way as TabSeparated.

## CSVWithNames

Also prints the header row, similar to [TabSeparatedWithNames](#).

## CustomSeparated

Similar to [Template](#), but it prints or reads all columns and uses escaping rule from setting `format_customEscapingRule` and delimiters from settings `formatCustomFieldDelimiter`, `formatCustomRowBeforeDelimiter`, `formatCustomRowAfterDelimiter`, `formatCustomRowBetweenDelimiter`, `formatCustomResultBeforeDelimiter` and `formatCustomResultAfterDelimiter`, not from format strings. There is also `CustomSeparatedIgnoreSpaces` format, which is similar to `TemplateIgnoreSpaces`.

## JSON

Outputs data in JSON format. Besides data tables, it also outputs column names and types, along with some additional information: the total number of output rows, and the number of rows that could have been output if there weren't a LIMIT. Example:

```
SELECT SearchPhrase, count() AS c FROM test.hits GROUP BY SearchPhrase WITH TOTALS ORDER BY c DESC LIMIT 5
FORMAT JSON
```

```
{
  "meta": [
    {
      "name": "hello",
      "type": "String"
    },
    {
      "name": "multiply(42, number)",
      "type": "UInt64"
    },
    {
      "name": "range(5)",
      "type": "Array(UInt8)"
    }
  ],
  "data": [
    {
      "hello": "hello",
      "multiply(42, number)": "0",
      "range(5)": [0,1,2,3,4]
    },
    {
      "hello": "hello",
      "multiply(42, number)": "42",
      "range(5)": [0,1,2,3,4]
    },
    {
      "hello": "hello",
      "multiply(42, number)": "84",
      "range(5)": [0,1,2,3,4]
    }
  ],
  "rows": 3,
  "rows_before_limit_at_least": 3
}
```

The JSON is compatible with JavaScript. To ensure this, some characters are additionally escaped: the slash / is escaped as \/; alternative line breaks U+2028 and U+2029, which break some browsers, are escaped as \uXXXX. ASCII control characters are escaped: backspace, form feed, line feed, carriage return, and horizontal tab are replaced with \b, \f, \n, \r, \t, as well as the remaining bytes in the 00-1F range using \uXXXX sequences. Invalid UTF-8 sequences are changed to the replacement character \uFFFD so the output text will consist of valid UTF-8 sequences. For compatibility with JavaScript, Int64 and UInt64 integers are enclosed in double-quotes by default. To remove the quotes, you can set the configuration parameter [output\\_format\\_json\\_quote\\_64bit\\_integers](#) to 0.

`rows` – The total number of output rows.

`rows_before_limit_at_least` The minimal number of rows there would have been without LIMIT. Output only if the query contains LIMIT.

If the query contains GROUP BY, `rows_before_limit_at_least` is the exact number of rows there would have been without a LIMIT.

`totals` – Total values (when using WITH TOTALS).

`extremes` – Extreme values (when extremes are set to 1).

This format is only appropriate for outputting a query result, but not for parsing (retrieving data to insert in a table).

ClickHouse supports `NULL`, which is displayed as `null` in the JSON output. To enable `+nan`, `-nan`, `+inf`, `-inf` values in output, set the [output\\_format\\_json\\_quote\\_denormals](#) to 1.

## See Also

- [JSONEachRow](#) format
- [output\\_format\\_json\\_array\\_of\\_rows](#) setting

## JSONStrings

Differs from JSON only in that data fields are output in strings, not in typed JSON values.

Example:

```
{
    "meta": [
        {
            "name": "'hello'",
            "type": "String"
        },
        {
            "name": "multiply(42, number)",
            "type": "UInt64"
        },
        {
            "name": "range(5)",
            "type": "Array(UInt8)"
        }
    ],
    "data": [
        {
            "'hello)": "hello",
            "multiply(42, number)": "0",
            "range(5)": "[0,1,2,3,4]"
        },
        {
            "'hello)": "hello",
            "multiply(42, number)": "42",
            "range(5)": "[0,1,2,3,4]"
        },
        {
            "'hello)": "hello",
            "multiply(42, number)": "84",
            "range(5)": "[0,1,2,3,4]"
        }
    ],
    "rows": 3,
    "rows_before_limit_at_least": 3
}
```

## JSONAsString

In this format, a single JSON object is interpreted as a single value. If the input has several JSON objects (comma separated), they are interpreted as separate rows. If the input data is enclosed in square brackets, it is interpreted as an array of JSONs.

This format can only be parsed for table with a single field of type [String](#). The remaining columns must be set to [DEFAULT](#) or [MATERIALIZED](#), or omitted. Once you collect whole JSON object to string you can use [JSON functions](#) to process it.

### Examples

Query:

```
DROP TABLE IF EXISTS json_as_string;
CREATE TABLE json_as_string (json String) ENGINE = Memory;
INSERT INTO json_as_string (json) FORMAT JSONAsString {"foo":{"bar":{"x":"y"}, "baz":1}, {}, {"any json stucture":1}
SELECT * FROM json_as_string;
```

Result:

```
json
{"foo": {"bar": {"x": "y"}, "baz": 1} } |
{} |
{"any json stucture": 1} |
```

## An array of JSON objects

Query:

```
CREATE TABLE json_square_brackets (field String) ENGINE = Memory;
INSERT INTO json_square_brackets FORMAT JSONAsString [{"id": 1, "name": "name1"}, {"id": 2, "name": "name2"}];

SELECT * FROM json_square_brackets;
```

Result:

```
field
{"id": 1, "name": "name1"} |
{"id": 2, "name": "name2"} |
```

## JSONCompact

### JSONCompactStrings

Differs from JSON only in that data rows are output in arrays, not in objects.

Example:

```
// JSONCompact
{
    "meta": [
        {
            "name": "'hello'",
            "type": "String"
        },
        {
            "name": "multiply(42, number)",
            "type": "UInt64"
        },
        {
            "name": "range(5)",
            "type": "Array(UInt8)"
        }
    ],
    "data": [
        ["hello", "0", [0,1,2,3,4]],
        ["hello", "42", [0,1,2,3,4]],
        ["hello", "84", [0,1,2,3,4]]
    ],
    "rows": 3,
    "rows_before_limit_at_least": 3
}
```

```
// JSONCompactStrings
{
    "meta": [
        {
            "name": "'hello'",
            "type": "String"
        },
        {
            "name": "multiply(42, number)",
            "type": "UInt64"
        },
        {
            "name": "range(5)",
            "type": "Array(UInt8)"
        }
    ],
    "data": [
        ["hello", "0", "[0,1,2,3,4]"],
        ["hello", "42", "[0,1,2,3,4]"],
        ["hello", "84", "[0,1,2,3,4]"]
    ],
    "rows": 3,
    "rows_before_limit_at_least": 3
}
```

## JSONEachRow

## JSONStringsEachRow

## JSONCompactEachRow

## JSONCompactStringsEachRow

When using these formats, ClickHouse outputs rows as separated, newline-delimited JSON values, but the data as a whole is not valid JSON.

```
{"some_int":42,"some_str":"hello","some_tuple":[1,"a"]} // JSONEachRow
[42,"hello",[1,"a"]] // JSONCompactEachRow
["42","hello","(2,'a')"] // JSONCompactStringsEachRow
```

When inserting the data, you should provide a separate JSON value for each row.

## JSONEachRowWithProgress

## JSONStringsEachRowWithProgress

Differs from `JSONEachRow/JSONStringsEachRow` in that ClickHouse will also yield progress information as JSON values.

```
{"row":{"'hello':'hello','multiply(42, number)':'0','range(5)':[0,1,2,3,4]}},
{"row":{"'hello':'hello','multiply(42, number)':'42','range(5)':[0,1,2,3,4]}},
 {"row":{"'hello':'hello','multiply(42, number)':'84','range(5)':[0,1,2,3,4]}},
 {"progress":{"read_rows":"3","read_bytes":"24","written_rows":"0","written_bytes":"0","total_rows_to_read":"3"}}
```

## JSONCompactEachRowWithNamesAndTypes

## JSONCompactStringsEachRowWithNamesAndTypes

Differs from `JSONCompactEachRow`/`JSONCompactStringsEachRow` in that the column names and types are written as the first two rows.

```
["hello", "multiply(42, number)", "range(5)"]
[\"String\", \"UInt64\", \"Array(UInt8)\"]
[\"hello\", \"0\", [0,1,2,3,4]]
[\"hello\", \"42\", [0,1,2,3,4]]
[\"hello\", \"84\", [0,1,2,3,4]]
```

## Inserting Data

```
INSERT INTO UserActivity FORMAT JSONEachRow {"PageViews":5, "UserID":"4324182021466249494", "Duration":146,"Sign":-1} {"UserID":"4324182021466249494", "PageViews":6,"Duration":185,"Sign":1}
```

ClickHouse allows:

- Any order of key-value pairs in the object.
- Omitting some values.

ClickHouse ignores spaces between elements and commas after the objects. You can pass all the objects in one line. You do not have to separate them with line breaks.

### Omitted values processing

ClickHouse substitutes omitted values with the default values for the corresponding [data types](#).

If `DEFAULT expr` is specified, ClickHouse uses different substitution rules depending on the `input_format_defaults_for_omitted_fields` setting.

Consider the following table:

```
CREATE TABLE IF NOT EXISTS example_table
(
    x UInt32,
    a DEFAULT x * 2
) ENGINE = Memory;
```

- If `input_format_defaults_for_omitted_fields = 0`, then the default value for `x` and `a` equals `0` (as the default value for the `UInt32` data type).
- If `input_format_defaults_for_omitted_fields = 1`, then the default value for `x` equals `0`, but the default value of `a` equals `x * 2`.

### Warning

When inserting data with `input_format_defaults_for_omitted_fields = 1`, ClickHouse consumes more computational resources, compared to insertion with `input_format_defaults_for_omitted_fields = 0`.

## Selecting Data

Consider the `UserActivity` table as an example:

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

The query `SELECT * FROM UserActivity FORMAT JSONEachRow` returns:

```
{"UserID":"4324182021466249494","PageViews":5,"Duration":146,"Sign":-1}  
{"UserID":"4324182021466249494","PageViews":6,"Duration":185,"Sign":1}
```

Unlike the [JSON](#) format, there is no substitution of invalid UTF-8 sequences. Values are escaped in the same way as for [JSON](#).

## Note

Any set of bytes can be output in the strings. Use the [JSONEachRow](#) format if you are sure that the data in the table can be formatted as [JSON](#) without losing any information.

## Usage of Nested Structures

If you have a table with [Nested](#) data type columns, you can insert [JSON](#) data with the same structure. Enable this feature with the [input\\_format\\_import\\_nested\\_json](#) setting.

For example, consider the following table:

```
CREATE TABLE json_each_row_nested (n Nested (s String, i Int32) ) ENGINE = Memory
```

As you can see in the [Nested](#) data type description, ClickHouse treats each component of the nested structure as a separate column (`n.s` and `n.i` for our table). You can insert data in the following way:

```
INSERT INTO json_each_row_nested FORMAT JSONEachRow {"n.s": ["abc", "def"], "n.i": [1, 23]}
```

To insert data as a hierarchical [JSON](#) object, set [input\\_format\\_import\\_nested\\_json=1](#).

```
{  
  "n": {  
    "s": ["abc", "def"],  
    "i": [1, 23]  
  }  
}
```

Without this setting, ClickHouse throws an exception.

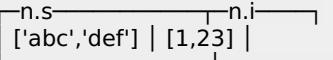
```
SELECT name, value FROM system.settings WHERE name = 'input_format_import_nested_json'
```

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
```



## Native

The most efficient format. Data is written and read by blocks in binary format. For each block, the number of rows, number of columns, column names and types, and parts of columns in this block are recorded one after another. In other words, this format is “columnar” – it does not convert columns to rows. This is the format used in the native interface for interaction between servers, for using the command-line client, and for C++ clients.

You can use this format to quickly generate dumps that can only be read by the ClickHouse DBMS. It does not make sense to work with this format yourself.

## Null

Nothing is output. However, the query is processed, and when using the command-line client, data is transmitted to the client. This is used for tests, including performance testing.

Obviously, this format is only appropriate for output, not for parsing.

## Pretty

Outputs data as Unicode-art tables, also using ANSI-escape sequences for setting colours in the terminal. A full grid of the table is drawn, and each row occupies two lines in the terminal.

Each result block is output as a separate table. This is necessary so that blocks can be output without buffering results (buffering would be necessary in order to pre-calculate the visible width of all the values).

**NULL** is output as `NULL`.

Example (shown for the **PrettyCompact** format):

```
SELECT * FROM t_null
```



Rows are not escaped in Pretty\* formats. Example is shown for the **PrettyCompact** format:

```
SELECT 'String with \'quotes\' and \t character' AS Escaping_test
```



To avoid dumping too much data to the terminal, only the first 10,000 rows are printed. If the number of rows is greater than or equal to 10,000, the message “Showed first 10 000” is printed.

This format is only appropriate for outputting a query result, but not for parsing (retrieving data to insert in a table).

The Pretty format supports outputting total values (when using WITH TOTALS) and extremes (when 'extremes' is set to 1). In these cases, total values and extreme values are output after the main data, in separate tables. Example (shown for the **PrettyCompact** format):

```
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
1970-01-01	8873898

Extremes:

EventDate	c
2014-03-17	1031592
2014-03-23	1406958

## PrettyCompact

Differs from **Pretty** in that the grid is drawn between rows and the result is more compact.

This format is used by default in the command-line client in interactive mode.

## PrettyCompactMonoBlock

Differs from **PrettyCompact** in that up to 10,000 rows are buffered, then output as a single table, not by blocks.

## PrettyNoEscapes

Differs from Pretty in that ANSI-escape sequences aren't used. This is necessary for displaying this format in a browser, as well as for using the 'watch' command-line utility.

Example:

```
$ watch -n1 "clickhouse-client --query='SELECT event, value FROM system.events FORMAT  
PrettyCompactNoEscapes'"
```

You can use the HTTP interface for displaying in the browser.

## PrettyCompactNoEscapes

The same as the previous setting.

## PrettySpaceNoEscapes

The same as the previous setting.

## PrettySpace

Differs from **PrettyCompact** in that whitespace (space characters) is used instead of the grid.

## RowBinary

Formats and parses data by row in binary format. Rows and values are listed consecutively, without separators.

This format is less efficient than the Native format since it is row-based.

Integers use fixed-length little-endian representation. For example, UInt64 uses 8 bytes.

DateTime is represented as UInt32 containing the Unix timestamp as the value.

Date is represented as a UInt16 object that contains the number of days since 1970-01-01 as the value.

String is represented as a varint length (unsigned LEB128), followed by the bytes of the string.

FixedString is represented simply as a sequence of bytes.

Array is represented as a varint length (unsigned LEB128), followed by successive elements of the array.

For **NULL** support, an additional byte containing 1 or 0 is added before each **Nullable** value. If 1, then the value is **NULL** and this byte is interpreted as a separate value. If 0, the value after the byte is not **NULL**.

## RowBinaryWithNamesAndTypes

Similar to [RowBinary](#), but with added header:

- LEB128-encoded number of columns (N)
- N Strings specifying column names
- N Strings specifying column types

## Values

Prints every row in brackets. Rows are separated by commas. There is no comma after the last row. The values inside the brackets are also comma-separated. Numbers are output in a decimal format without quotes. Arrays are output in square brackets. Strings, dates, and dates with times are output in quotes. Escaping rules and parsing are similar to the [TabSeparated](#) format. During formatting, extra spaces aren't inserted, but during parsing, they are allowed and skipped (except for spaces inside array values, which are not allowed). **NULL** is represented as **NULL**.

The minimum set of characters that you need to escape when passing data in Values format: single quotes and backslashes.

This is the format that is used in `INSERT INTO t VALUES ...`, but you can also use it for formatting query results.

See also: [input\\_format\\_values\\_interpret\\_expressions](#) and  
[input\\_format\\_values\\_deduce\\_templates\\_of\\_expressions](#) settings.

## Vertical

Prints each value on a separate line with the column name specified. This format is convenient for printing just one or a few rows if each row consists of a large number of columns.

**NULL** is output as **NULL**.

Example:

```
SELECT * FROM t_null FORMAT Vertical
```

Row 1:

x: 1  
y: NULL

Rows are not escaped in Vertical format:

```
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  
characters

This format is only appropriate for outputting a query result, but not for parsing (retrieving data to insert in a table).

## XML

XML format is suitable only for output, not for parsing. Example:

```

<?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>
        <row>
            <SearchPhrase>baku</SearchPhrase>
            <field>1000</field>
        </row>
    </data>
    <rows>10</rows>
    <rows_before_limit_at_least>141137</rows_before_limit_at_least>
</result>

```

If the column name does not have an acceptable format, just ‘field’ is used as the element name. In general, the XML structure follows the JSON structure.

Just as for JSON, invalid UTF-8 sequences are changed to the replacement character ♦ so the output text will consist of valid UTF-8 sequences.

In string values, the characters < and & are escaped as < and &.

Arrays are output as <array><elem>Hello</elem><elem>World</elem>...</array>, and tuples as <tuple><elem>Hello</elem><elem>World</elem>...</tuple>.

## CapnProto

Cap'n Proto is a binary message format similar to Protocol Buffers and Thrift, but not like JSON or MessagePack.

Cap'n Proto messages are strictly typed and not self-describing, meaning they need an external schema description. The schema is applied on the fly and cached for each query.

```
$ cat capnproto_messages.bin | clickhouse-client --query "INSERT INTO test.hits FORMAT CapnProto SETTINGS format_schema='schema:Message'"
```

Where schema.capnp looks like this:

```
struct Message {  
    SearchPhrase @0 :Text;  
    c @1 :UInt64;  
}
```

Deserialization is effective and usually does not increase the system load.

See also [Format Schema](#).

## Protobuf

Protobuf - is a [Protocol Buffers](#) format.

This format requires an external format schema. The schema is cached between queries.

ClickHouse supports both `proto2` and `proto3` syntaxes. Repeated/optional/required fields are supported.

Usage examples:

```
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'"
```

where the file `schemafile.proto` looks like this:

```
syntax = "proto3";  
  
message MessageType {  
    string name = 1;  
    string surname = 2;  
    uint32 birthDate = 3;  
    repeated string phoneNumbers = 4;  
};
```

To find the correspondence between table columns and fields of Protocol Buffers' message type ClickHouse compares their names.

This comparison is case-insensitive and the characters `_` (underscore) and `.` (dot) are considered as equal. If types of a column and a field of Protocol Buffers' message are different the necessary conversion is applied.

Nested messages are supported. For example, for the field `z` in the following message type

```

message MessageType {
    message XType {
        message YType {
            int32 z;
        };
        repeated YType y;
    };
    XType x;
}

```

ClickHouse tries to find a column named `x.y.z` (or `x_y_z` or `X.y_Z` and so on).

Nested messages are suitable to input or output a [nested data structures](#).

Default values defined in a protobuf schema like this

```

syntax = "proto2";

message MessageType {
    optional int32 result_per_page = 3 [default = 10];
}

```

are not applied; the [table defaults](#) are used instead of them.

ClickHouse inputs and outputs protobuf messages in the `length-delimited` format.

It means before every message should be written its length as a `varint`.

See also [how to read/write length-delimited protobuf messages in popular languages](#).

## ProtobufSingle

Same as [Protobuf](#) but for storing/parsing single Protobuf message without length delimiters.

## Avro

[Apache Avro](#) is a row-oriented data serialization framework developed within Apache's Hadoop project.

ClickHouse Avro format supports reading and writing [Avro data files](#).

## Data Types Matching

The table below shows supported data types and how they match ClickHouse [data types](#) in `INSERT` and `SELECT` queries.

Avro data type <code>INSERT</code>	ClickHouse data type	Avro data type <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>
<code>boolean, int, long, float, double</code>	<code>Float64</code>	<code>double</code>
<code>bytes, string, fixed, enum</code>	<code>String</code>	<code>bytes or string *</code>
<code>bytes, string, fixed</code>	<code>FixedString(N)</code>	<code>fixed(N)</code>
<code>enum</code>	<code>Enum(8 16)</code>	<code>enum</code>

Avro data type <code>INSERT</code>	ClickHouse data type	Avro data type <code>SELECT</code>
<code>array(T)</code>	<code>Array(T)</code>	<code>array(T)</code>
<code>union(null, T), union(T, null)</code>	<code>Nullable(T)</code>	<code>union(null, T)</code>
<code>null</code>	<code>Nullable(Nothing)</code>	<code>null</code>
<code>int (date) **</code>	<code>Date</code>	<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>

\* bytes is default, controlled by [output\\_format\\_avro\\_string\\_column\\_pattern](#)

\*\* Avro logical types

Unsupported Avro data types: `record` (non-root), `map`

Unsupported Avro logical data types: `time-millis`, `time-micros`, `duration`

## Inserting Data

To insert data from an Avro file into ClickHouse table:

```
$ cat file.avro | clickhouse-client --query="INSERT INTO {some_table} FORMAT Avro"
```

The root schema of input Avro file must be of `record` type.

To find the correspondence between table columns and fields of Avro schema ClickHouse compares their names. This comparison is case-sensitive.

Unused fields are skipped.

Data types of ClickHouse table columns can differ from the corresponding fields of the Avro data inserted. When inserting data, ClickHouse interprets data types according to the table above and then [casts](#) the data to corresponding column type.

## Selecting Data

To select data from ClickHouse table into an Avro file:

```
$ clickhouse-client --query="SELECT * FROM {some_table} FORMAT Avro" > file.avro
```

Column names must:

- start with `[A-Za-z_]`
- subsequently contain only `[A-Za-z0-9_]`

Output Avro file compression and sync interval can be configured with [output\\_format\\_avro\\_codec](#) and [output\\_format\\_avro\\_sync\\_interval](#) respectively.

## AvroConfluent

AvroConfluent supports decoding single-object Avro messages commonly used with [Kafka](#) and [Confluent Schema Registry](#).

Each Avro message embeds a schema id that can be resolved to the actual schema with help of the Schema Registry.

Schemas are cached once resolved.

Schema Registry URL is configured with `format_avro_schema_registry_url`.

## Data Types Matching

Same as [Avro](#).

## Usage

To quickly verify schema resolution you can use [kafkacat](#) with [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
```

To use [AvroConfluent](#) with [Kafka](#):

```
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;
```

## Warning

Setting `format_avro_schema_registry_url` needs to be configured in `users.xml` to maintain its value after a restart. Also you can use the `format_avro_schema_registry_url` setting of the [Kafka](#) table engine.

## Parquet

[Apache Parquet](#) is a columnar storage format widespread in the Hadoop ecosystem. ClickHouse supports read and write operations for this format.

## Data Types Matching

The table below shows supported data types and how they match ClickHouse [data types](#) in `INSERT` and `SELECT` queries.

Parquet data type ( <code>INSERT</code> )	ClickHouse data type	Parquet data type ( <code>SELECT</code> )
<code>UINT8, BOOL</code>	<code>UInt8</code>	<code>UINT8</code>

Parquet data type ( <code>INSERT</code> )	ClickHouse data type	Parquet data type ( <code>SELECT</code> )
INT8	Int8	INT8
UINT16	UInt16	UINT16
INT16	Int16	INT16
UINT32	UInt32	UINT32
INT32	Int32	INT32
UINT64	UInt64	UINT64
INT64	Int64	INT64
FLOAT, HALF_FLOAT	Float32	FLOAT
DOUBLE	Float64	DOUBLE
DATE32	Date	UINT16
DATE64, TIMESTAMP	DateTime	UINT32
STRING, BINARY	String	BINARY
—	FixedString	BINARY
DECIMAL	Decimal	DECIMAL
LIST	Array	LIST
STRUCT	Tuple	STRUCT
MAP	Map	MAP

Arrays can be nested and can have a value of the `Nullable` type as an argument. `Tuple` and `Map` types also can be nested.

ClickHouse supports configurable precision of `Decimal` type. The `INSERT` query treats the Parquet `DECIMAL` type as the ClickHouse `Decimal128` type.

Unsupported Parquet data types: `TIME32`, `FIXED_SIZE_BINARY`, `JSON`, `UUID`, `ENUM`.

Data types of ClickHouse table columns can differ from the corresponding fields of the Parquet data inserted. When inserting data, ClickHouse interprets data types according to the table above and then `cast` the data to that data type which is set for the ClickHouse table column.

## Inserting and Selecting Data

You can insert Parquet data from a file into ClickHouse table by the following command:

```
$ cat {filename} | clickhouse-client --query="INSERT INTO {some_table} FORMAT Parquet"
```

To insert data into **Nested** columns as an array of structs values you must switch on the `input_format_parquet_import_nested` setting.

You can select data from a ClickHouse table and save them into some file in the Parquet format by the following command:

```
$ clickhouse-client --query="SELECT * FROM {some_table} FORMAT Parquet" > {some_file.pq}
```

To exchange data with Hadoop, you can use **HDFS table engine**.

## Arrow

**Apache Arrow** comes with two built-in columnar storage formats. ClickHouse supports read and write operations for these formats.

**Arrow** is Apache Arrow's "file mode" format. It is designed for in-memory random access.

## Data Types Matching

The table below shows supported data types and how they match ClickHouse **data types** in `INSERT` and `SELECT` queries.

Arrow data type ( <code>INSERT</code> )	ClickHouse data type	Arrow data type ( <code>SELECT</code> )
UINT8, BOOL	UInt8	UINT8
INT8	Int8	INT8
UINT16	UInt16	UINT16
INT16	Int16	INT16
UINT32	UInt32	UINT32
INT32	Int32	INT32
UINT64	UInt64	UINT64
INT64	Int64	INT64
FLOAT, HALF_FLOAT	Float32	FLOAT32
DOUBLE	Float64	FLOAT64
DATE32	Date	UINT16
DATE64, TIMESTAMP	DateTime	UINT32
STRING, BINARY	String	BINARY
STRING, BINARY	FixedString	BINARY
DECIMAL	Decimal	DECIMAL

Arrow data type ( <code>INSERT</code> )	ClickHouse data type	Arrow data type ( <code>SELECT</code> )
DECIMAL256	Decimal256	DECIMAL256
LIST	Array	LIST
STRUCT	Tuple	STRUCT
MAP	Map	MAP

Arrays can be nested and can have a value of the `Nullable` type as an argument. `Tuple` and `Map` types also can be nested.

The `DICTIONARY` type is supported for `INSERT` queries, and for `SELECT` queries there is an `output_format_arrow_low_cardinality_as_dictionary` setting that allows to output `LowCardinality` type as a `DICTIONARY` type.

ClickHouse supports configurable precision of the `Decimal` type. The `INSERT` query treats the Arrow `DECIMAL` type as the ClickHouse `Decimal128` type.

Unsupported Arrow data types: `TIME32`, `FIXED_SIZE_BINARY`, `JSON`, `UUID`, `ENUM`.

The data types of ClickHouse table columns do not have to match the corresponding Arrow data fields. When inserting data, ClickHouse interprets data types according to the table above and then `casts` the data to the data type set for the ClickHouse table column.

## Inserting Data

You can insert Arrow data from a file into ClickHouse table by the following command:

```
$ cat filename.arrow | clickhouse-client --query="INSERT INTO some_table FORMAT Arrow"
```

To insert data into `Nested` columns as an array of structs values you must switch on the `input_format_arrow_import_nested` setting.

## Selecting Data

You can select data from a ClickHouse table and save them into some file in the Arrow format by the following command:

```
$ clickhouse-client --query="SELECT * FROM {some_table} FORMAT Arrow" > {filename.arrow}
```

## ArrowStream

`ArrowStream` is Apache Arrow's "stream mode" format. It is designed for in-memory stream processing.

## ORC

Apache ORC is a columnar storage format widespread in the `Hadoop` ecosystem.

## Data Types Matching

The table below shows supported data types and how they match ClickHouse `data types` in `INSERT` and `SELECT` queries.

ORC data type ( <code>INSERT</code> )	ClickHouse data type	ORC data type ( <code>SELECT</code> )
UINT8, BOOL	UInt8	UINT8
INT8	Int8	INT8
UINT16	UInt16	UINT16
INT16	Int16	INT16
UINT32	UInt32	UINT32
INT32	Int32	INT32
UINT64	UInt64	UINT64
INT64	Int64	INT64
FLOAT, HALF_FLOAT	Float32	FLOAT
DOUBLE	Float64	DOUBLE
DATE32	Date	DATE32
DATE64, TIMESTAMP	DateTime	TIMESTAMP
STRING, BINARY	String	BINARY
DECIMAL	Decimal	DECIMAL
LIST	Array	LIST
STRUCT	Tuple	STRUCT
MAP	Map	MAP

Arrays can be nested and can have a value of the `Nullable` type as an argument. `Tuple` and `Map` types also can be nested.

ClickHouse supports configurable precision of the `Decimal` type. The `INSERT` query treats the ORC `DECIMAL` type as the ClickHouse `Decimal128` type.

Unsupported ORC data types: `TIME32`, `FIXED_SIZE_BINARY`, `JSON`, `UUID`, `ENUM`.

The data types of ClickHouse table columns do not have to match the corresponding ORC data fields. When inserting data, ClickHouse interprets data types according to the table above and then `casts` the data to the data type set for the ClickHouse table column.

## Inserting Data

You can insert ORC data from a file into ClickHouse table by the following command:

```
$ cat filename.orc | clickhouse-client --query="INSERT INTO some_table FORMAT ORC"
```

To insert data into [Nested](#) columns as an array of structs values you must switch on the `input_format_orc_import_nested` setting.

## Selecting Data

You can select data from a ClickHouse table and save them into some file in the ORC format by the following command:

```
$ clickhouse-client --query="SELECT * FROM {some_table} FORMAT ORC" > {filename.orc}
```

To exchange data with Hadoop, you can use [HDFS table engine](#).

## LineAsString

In this format, every line of input data is interpreted as a single string value. This format can only be parsed for table with a single field of type [String](#). The remaining columns must be set to [DEFAULT](#) or [MATERIALIZED](#), or omitted.

### Example

Query:

```
DROP TABLE IF EXISTS line_as_string;
CREATE TABLE line_as_string (field String) ENGINE = Memory;
INSERT INTO line_as_string FORMAT LineAsString "I love apple", "I love banana", "I love orange";
SELECT * FROM line_as_string;
```

Result:

```
field
"I love apple", "I love banana", "I love orange"; |
```

## Regexp

Each line of imported data is parsed according to the regular expression.

When working with the [Regexp](#) format, you can use the following settings:

- `format_regex` — [String](#). Contains regular expression in the [re2](#) format.
- `format_regex_escaping_rule` — [String](#). The following escaping rules are supported:
  - CSV (similarly to [CSV](#))
  - JSON (similarly to [JSONEachRow](#))
  - Escaped (similarly to [TSV](#))
  - Quoted (similarly to [Values](#))
  - Raw (extracts subpatterns as a whole, no escaping rules)
- `format_regex_skip_unmatched` — [UInt8](#). Defines the need to throw an exception in case the `format_regex` expression does not match the imported data. Can be set to 0 or 1.

### Usage

The regular expression from `format_regex` setting is applied to every line of imported data. The number of subpatterns in the regular expression must be equal to the number of columns in imported dataset.

Lines of the imported data must be separated by newline character '\n' or DOS-style newline "\r\n".

The content of every matched subpattern is parsed with the method of corresponding data type, according to `format_regex_escaping_rule` setting.

If the regular expression does not match the line and `format_regex_skip_unmatched` is set to 1, the line is silently skipped. If `format_regex_skip_unmatched` is set to 0, exception is thrown.

## Example

Consider the file `data.tsv`:

```
id: 1 array: [1,2,3] string: str1 date: 2020-01-01
id: 2 array: [1,2,3] string: str2 date: 2020-01-02
id: 3 array: [1,2,3] string: str3 date: 2020-01-03
```

and the table:

```
CREATE TABLE imp_regex_table (id UInt32, array Array(UInt32), string String, date Date) ENGINE = Memory;
```

Import command:

```
$ cat data.tsv | clickhouse-client --query "INSERT INTO imp_regex_table FORMAT Regexp SETTINGS
format_regex='id: (.+?) array: (.+?) string: (.+?) date: (.+?)', format_regex_escaping_rule='Escaped',
format_regex_skip_unmatched=0;"
```

Query:

```
SELECT * FROM imp_regex_table;
```

Result:

id	array	string	date
1	[1,2,3]	str1	2020-01-01
2	[1,2,3]	str2	2020-01-02
3	[1,2,3]	str3	2020-01-03

## Format Schema

The file name containing the format schema is set by the setting `format_schema`.

It's required to set this setting when it is used one of the formats `Cap'n Proto` and `Protobuf`.

The format schema is a combination of a file name and the name of a message type in this file, delimited by a colon,

e.g. `schemafile.proto:MessageType`.

If the file has the standard extension for the format (for example, `.proto` for `Protobuf`), it can be omitted and in this case, the format schema looks like `schemafile:MessageType`.

If you input or output data via the `client` in the [interactive mode](#), the file name specified in the format schema

can contain an absolute path or a path relative to the current directory on the client.

If you use the client in the [batch mode](#), the path to the schema must be relative due to security reasons.

If you input or output data via the [HTTP interface](#) the file name specified in the format schema should be located in the directory specified in `format_schema_path` in the server configuration.

# Skipping Errors

Some formats such as `CSV`, `TabSeparated`, `TSKV`, `JSONEachRow`, `Template`, `CustomSeparated` and `Protobuf` can skip broken row if parsing error occurred and continue parsing from the beginning of next row. See `input_format_allow_errors_num` and `input_format_allow_errors_ratio` settings.

Limitations:

- In case of parsing error `JSONEachRow` skips all data until the new line (or EOF), so rows must be delimited by `\n` to count errors correctly.
- `Template` and `CustomSeparated` use delimiter after the last column and delimiter between rows to find the beginning of next row, so skipping errors works only if at least one of them is not empty.

# RawBLOB

In this format, all input data is read to a single value. It is possible to parse only a table with a single field of type `String` or similar.

The result is output in binary format without delimiters and escaping. If more than one value is output, the format is ambiguous, and it will be impossible to read the data back.

Below is a comparison of the formats `RawBLOB` and `TabSeparatedRaw`.

`RawBLOB`:

- data is output in binary format, no escaping;
  - there are no delimiters between values;
  - no newline at the end of each value.
- [`TabSeparatedRaw`] (#tabseparatedraw):
- data is output without escaping;
  - the rows contain values separated by tabs;
  - there is a line feed after the last value in every row.

The following is a comparison of the `RawBLOB` and `RowBinary` formats.

`RawBLOB`:

- String fields are output without being prefixed by length.

`RowBinary`:

- String fields are represented as length in varint format (unsigned [LEB128] (<https://en.wikipedia.org/wiki/LEB128>)), followed by the bytes of the string.

When an empty data is passed to the `RawBLOB` input, ClickHouse throws an exception:

```
Code: 108. DB::Exception: No data to insert
```

## Example

```
$ clickhouse-client --query "CREATE TABLE {some_table} (a String) ENGINE = Memory;"  
$ cat {filename} | clickhouse-client --query="INSERT INTO {some_table} FORMAT RawBLOB"  
$ clickhouse-client --query "SELECT * FROM {some_table} FORMAT RawBLOB" | md5sum
```

Result:

```
f9725a22f9191e064120d718e26862a9 -
```

# MsgPack

ClickHouse supports reading and writing `MessagePack` data files.

## Data Types Matching

MessagePack data type ( <a href="#">INSERT</a> )	ClickHouse data type	MessagePack data type ( <a href="#">SELECT</a> )
uint N, positive fixint	UIntN	uint N
int N	IntN	int N
bool	UInt8	uint 8
fixstr, str 8, str 16, str 32, bin 8, bin 16, bin 32	String	bin 8, bin 16, bin 32
fixstr, str 8, str 16, str 32, bin 8, bin 16, bin 32	FixedString	bin 8, bin 16, bin 32
float 32	Float32	float 32
float 64	Float64	float 64
uint 16	Date	uint 16
uint 32	DateTime	uint 32
uint 64	DateTime64	uint 64
fixarray, array 16, array 32	Array	fixarray, array 16, array 32
fixmap, map 16, map 32	Map	fixmap, map 16, map 32

Example:

Writing to a file ".msgpk":

```
$ clickhouse-client --query="CREATE TABLE msgpack (array Array(UInt8)) ENGINE = Memory;"$ clickhouse-client --query="INSERT INTO msgpack VALUES ([0, 1, 2, 3, 42, 253, 254, 255]), ([255, 254, 253, 42, 3, 2, 1, 0]);"$ clickhouse-client --query="SELECT * FROM msgpack FORMAT MsgPack" > tmp_msgpack.msgpk;
```

## JDBC Driver

- [Official driver](#)
- Third-party drivers:
  - [ClickHouse-Native-JDBC](#)
  - [clickhouse4j](#)

## ODBC Driver

- [Official driver](#)

## C++ Client Library

See README at [clickhouse-cpp](#) repository.

## Third-Party Interfaces

This is a collection of links to third-party tools that provide some sort of interface to ClickHouse. It can be either visual interface, command-line interface or an API:

- [Client libraries](#)
- [Integrations](#)
- [GUI](#)
- [Proxies](#)

### Note

Generic tools that support common API like [ODBC](#) or [JDBC](#) usually can work with ClickHouse as well, but are not listed here because there are way too many of them.

## Client Libraries from Third-party Developers

### Disclaimer

Yandex does **not** maintain the libraries listed below and hasn't done any extensive testing to ensure their quality.

- Python
  - [infi.clickhouse\\_orm](#)
  - [clickhouse-driver](#)
  - [clickhouse-client](#)
  - [aiochclient](#)
  - [asynch](#)

- PHP
  - [smi2/phpclickhouse](#)
  - [8bitov/clickhouse-php-client](#)
  - [bozerkins/clickhouse-client](#)
  - [simpod/clickhouse-client](#)
  - [seva-code/php-click-house-client](#)
  - [SeasClick C++ client](#)
  - [one-ck](#)
  - [glushkovds/phpclickhouse-laravel](#)
  - [kolya7k ClickHouse PHP extension](#)
- Go
  - [clickhouse](#)
  - [go-clickhouse](#)
  - [mailrugo-clickhouse](#)
  - [golang-clickhouse](#)
- Swift
  - [ClickHouseNIO](#)
  - [ClickHouseVapor ORM](#)
- NodeJs
  - [clickhouse \(NodeJs\)](#)
  - [node-clickhouse](#)
- Perl
  - [perl-DBD-ClickHouse](#)
  - [HTTP-ClickHouse](#)
  - [AnyEvent-ClickHouse](#)
- Ruby
  - [ClickHouse \(Ruby\)](#)
  - [clickhouse-activerecord](#)
- Rust
  - [Klickhouse](#)
- R
  - [clickhouse-r](#)
  - [RClickHouse](#)
- Java
  - [clickhouse-client-java](#)
  - [clickhouse-client](#)

- Scala
  - [clickhouse-scala-client](#)
- Kotlin
  - [AORM](#)
- C#
  - [Octonica.ClickHouseClient](#)
  - [ClickHouse.Ado](#)
  - [ClickHouse.Client](#)
  - [ClickHouse.Net](#)
- Elixir
  - [clickhousex](#)
  - [pillar](#)
- Nim
  - [nim-clickhouse](#)
- Haskell
  - [hdbc-clickhouse](#)

## Integration Libraries from Third-party Developers

### Disclaimer

Yandex does **not** maintain the tools and libraries listed below and haven't done any extensive testing to ensure their quality.

## Infrastructure Products

- Relational database management systems
  - MySQL
    - [mysql2ch](#)
    - [ProxySQL](#)
    - [clickhouse-mysql-data-reader](#)
    - [horgh-replicator](#)
  - PostgreSQL
    - [clickhousedb\\_fdw](#)
    - [infi.clickhouse\\_fdw](#) (uses [infi.clickhouse\\_orm](#))
    - [pg2ch](#)
    - [clickhouse\\_fdw](#)
  - MSSQL
    - [ClickHouseMigrator](#)

- Message queues
  - [Kafka](#)
    - [clickhouse\\_sinker](#) (uses [Go client](#))
    - [stream-loader-clickhouse](#)
- Stream processing
  - [Flink](#)
    - [flink-clickhouse-sink](#)
- Object storages
  - [S3](#)
    - [clickhouse-backup](#)
- Container orchestration
  - [Kubernetes](#)
    - [clickhouse-operator](#)
- Configuration management
  - [puppet](#)
    - [innogames/clickhouse](#)
    - [mfedorov/clickhouse](#)
- Monitoring
  - [Graphite](#)
    - [graphouse](#)
    - [carbon-clickhouse](#)
    - [graphite-clickhouse](#)
  - [graphite-ch-optimizer](#) - optimizes stale partitions in [\\*GraphiteMergeTree](#) if rules from [rollup configuration](#) could be applied
  - [Grafana](#)
    - [clickhouse-grafana](#)
  - [Prometheus](#)
    - [clickhouse\\_exporter](#)
    - [PromHouse](#)
    - [clickhouse\\_exporter](#) (uses [Go client](#))
  - [Nagios](#)
    - [check\\_clickhouse](#)
    - [check\\_clickhouse.py](#)
  - [Zabbix](#)
    - [clickhouse-zabbix-template](#)
  - [Semantext](#)
    - [clickhouse integration](#)

- Logging
  - [rsyslog](#)
    - [omclickhouse](#)
  - [fluentd](#)
    - [loghouse](#) (for [Kubernetes](#))
  - [logagent](#)
    - [logagent output-plugin-clickhouse](#)
- Geo
  - [MaxMind](#)
    - [clickhouse-maxmind-geoip](#)
- AutoML
  - [MindsDB](#)
    - [MindsDB](#) - Predictive AI layer for ClickHouse database.

## Programming Language Ecosystems

- Python
  - [SQLAlchemy](#)
    - [sqlalchemy-clickhouse](#) (uses [infi.clickhouse\\_orm](#))
  - [pandas](#)
    - [pandahouse](#)
- PHP
  - [Doctrine](#)
    - [dbal-clickhouse](#)
- R
  - [dplyr](#)
    - [RClickHouse](#) (uses [clickhouse-cpp](#))
- Java
  - [Hadoop](#)
    - [clickhouse-hdfs-loader](#) (uses [JDBC](#))
- Scala
  - [Akka](#)
    - [clickhouse-scala-client](#)
- C#
  - [ADO.NET](#)
    - [ClickHouse.Ado](#)
    - [ClickHouse.Client](#)
    - [ClickHouse.Net](#)
    - [ClickHouse.Net.Migrations](#)
- Elixir
  - [Ecto](#)
    - [clickhouse\\_ecto](#)

- Ruby
    - [Ruby on Rails](#)
      - [activecube](#)
      - [ActiveRecord](#)
    - [GraphQL](#)
      - [activecube-graphql](#)
- 

## Visual Interfaces from Third-party Developers

### Open-Source

#### Tabix

Web interface for ClickHouse in the [Tabix](#) project.

Features:

- Works with ClickHouse directly from the browser, without the need to install additional software.
- Query editor with syntax highlighting.
- Auto-completion of commands.
- Tools for graphical analysis of query execution.
- Colour scheme options.

[Tabix documentation](#).

#### HouseOps

[HouseOps](#) is a UI/IDE for OSX, Linux and Windows.

Features:

- Query builder with syntax highlighting. View the response in a table or JSON view.
- Export query results as CSV or JSON.
- List of processes with descriptions. Write mode. Ability to stop (**KILL**) a process.
- Database graph. Shows all tables and their columns with additional information.
- A quick view of the column size.
- Server configuration.

The following features are planned for development:

- Database management.
- User management.
- Real-time data analysis.
- Cluster monitoring.
- Cluster management.
- Monitoring replicated and Kafka tables.

## LightHouse

[LightHouse](#) is a lightweight web interface for ClickHouse.

Features:

- Table list with filtering and metadata.
- Table preview with filtering and sorting.
- Read-only queries execution.

## Redash

[Redash](#) is a platform for data visualization.

Supports for multiple data sources including ClickHouse, Redash can join results of queries from different data sources into one final dataset.

Features:

- Powerful editor of queries.
- Database explorer.
- Visualization tools, that allow you to represent data in different forms.

## Grafana

[Grafana](#) is a platform for monitoring and visualization.

"Grafana allows you to query, visualize, alert on and understand your metrics no matter where they are stored. Create, explore, and share dashboards with your team and foster a data driven culture. Trusted and loved by the community" — [grafana.com](#).

ClickHouse datasource plugin provides a support for ClickHouse as a backend database.

## DBeaver

[DBeaver](#) - universal desktop database client with ClickHouse support.

Features:

- Query development with syntax highlight and autocompletion.
- Table list with filters and metadata search.
- Table data preview.
- Full-text search.

By default, DBeaver does not connect using a session (the CLI for example does). If you require session support (for example to set settings for your session), edit the driver connection properties and set `session_id` to a random string (it uses the http connection under the hood). Then you can use any setting from the query window.

## clickhouse-cli

[clickhouse-cli](#) is an alternative command-line client for ClickHouse, written in Python 3.

Features:

- Autocompletion.

- Syntax highlighting for the queries and data output.
- Pager support for the data output.
- Custom PostgreSQL-like commands.

## clickhouse-flamegraph

[clickhouse-flamegraph](#) is a specialized tool to visualize the `system.trace_log` as [flamegraph](#).

## clickhouse-plantuml

[clickhouse-plantuml](#) is a script to generate [PlantUML](#) diagram of tables' schemes.

## xeus-clickhouse

[xeus-clickhouse](#) is a Jupyter kernel for ClickHouse, which supports query CH data using SQL in Jupyter.

## MindsDB Studio

[MindsDB](#) is an open-source AI layer for databases including ClickHouse that allows you to effortlessly develop, train and deploy state-of-the-art machine learning models. MindsDB Studio(GUI) allows you to train new models from database, interpret predictions made by the model, identify potential data biases, and evaluate and visualize model accuracy using the Explainable AI function to adapt and tune your Machine Learning models faster.

## DBM

[DBM](#) DBM is a visual management tool for ClickHouse!

Features:

- Support query history (pagination, clear all, etc.)
- Support selected sql clauses query
- Support terminating query
- Support table management (metadata, delete, preview)
- Support database management (delete, create)
- Support custom query
- Support multiple data sources management(connection test, monitoring)
- Support monitor (processor, connection, query)
- Support migrate data

## Bytebase

[Bytebase](#) is a web-based, open source schema change and version control tool for teams. It supports various databases including ClickHouse.

Features:

- Schema review between developers and DBAs.
- Database-as-Code, version control the schema in VCS such GitLab and trigger the deployment upon code commit.
- Streamlined deployment with per-environment policy.

- Full migration history.
- Schema drift detection.
- Backup and restore.
- RBAC.

## Commercial

### DataGrip

[DataGrip](#) is a database IDE from JetBrains with dedicated support for ClickHouse. It is also embedded in other IntelliJ-based tools: PyCharm, IntelliJ IDEA, GoLand, PhpStorm and others.

Features:

- Very fast code completion.
- ClickHouse syntax highlighting.
- Support for features specific to ClickHouse, for example, nested columns, table engines.
- Data Editor.
- Refactorings.
- Search and Navigation.

### Yandex DataLens

[Yandex DataLens](#) is a service of data visualization and analytics.

Features:

- Wide range of available visualizations, from simple bar charts to complex dashboards.
- Dashboards could be made publicly available.
- Support for multiple data sources including ClickHouse.
- Storage for materialized data based on ClickHouse.

DataLens is [available for free](#) for low-load projects, even for commercial use.

- [DataLens documentation](#).
- [Tutorial](#) on visualizing data from a ClickHouse database.

### Holistics Software

[Holistics](#) is a full-stack data platform and business intelligence tool.

Features:

- Automated email, Slack and Google Sheet schedules of reports.
- SQL editor with visualizations, version control, auto-completion, reusable query components and dynamic filters.
- Embedded analytics of reports and dashboards via iframe.
- Data preparation and ETL capabilities.

- SQL data modelling support for relational mapping of data.

## Looker

[Looker](#) is a data platform and business intelligence tool with support for 50+ database dialects including ClickHouse. Looker is available as a SaaS platform and self-hosted. Users can use Looker via the browser to explore data, build visualizations and dashboards, schedule reports, and share their insights with colleagues. Looker provides a rich set of tools to embed these features in other applications, and an API to integrate data with other applications.

Features:

- Easy and agile development using LookML, a language which supports curated [Data Modeling](#) to support report writers and end-users.
- Powerful workflow integration via Looker's [Data Actions](#).

[How to configure ClickHouse in Looker.](#)

## SeekTable

[SeekTable](#) is a self-service BI tool for data exploration and operational reporting. It is available both as a cloud service and a self-hosted version. Reports from SeekTable may be embedded into any web-app.

Features:

- Business users-friendly reports builder.
- Powerful report parameters for SQL filtering and report-specific query customizations.
- Can connect to ClickHouse both with a native TCP/IP endpoint and a HTTP(S) interface (2 different drivers).
- It is possible to use all power of ClickHouse SQL dialect in dimensions/measures definitions.
- [Web API](#) for automated reports generation.
- Supports reports development flow with account data [backup/restore](#); data models (cubes) / reports configuration is a human-readable XML and can be stored under version control system.

SeekTable is [free](#) for personal/individual usage.

[How to configure ClickHouse connection in SeekTable.](#)

## Chadmin

[Chadmin](#) is a simple UI where you can visualize your currently running queries on your ClickHouse cluster and info about them and kill them if you want.

## Proxy Servers from Third-party Developers

### chproxy

[chproxy](#), is an HTTP proxy and load balancer for ClickHouse database.

Features:

- Per-user routing and response caching.
- Flexible limits.

- Automatic SSL certificate renewal.

Implemented in Go.

## KittenHouse

[KittenHouse](#) is designed to be a local proxy between ClickHouse and application server in case it's impossible or inconvenient to buffer INSERT data on your application side.

Features:

- In-memory and on-disk data buffering.
- Per-table routing.
- Load-balancing and health checking.

Implemented in Go.

## ClickHouse-Bulk

[ClickHouse-Bulk](#) is a simple ClickHouse insert collector.

Features:

- Group requests and send by threshold or interval.
- Multiple remote servers.
- Basic authentication.

Implemented in Go.

---

## ClickHouse Engines

There are two key engine kinds in ClickHouse:

- [Table engines](#)
- [Database engines](#)

---

## Table Engines for Integrations

ClickHouse provides various means for integrating with external systems, including table engines. Like with all other table engines, the configuration is done using CREATE TABLE or ALTER TABLE queries. Then from a user perspective, the configured integration looks like a normal table, but queries to it are proxied to the external system. This transparent querying is one of the key advantages of this approach over alternative integration methods, like external dictionaries or table functions, which require to use custom query methods on each use.

List of supported integrations:

- [ODBC](#)
- [JDBC](#)
- [MySQL](#)
- [MongoDB](#)

- [HDFS](#)
- [S3](#)
- [Kafka](#)
- [EmbeddedRocksDB](#)
- [RabbitMQ](#)
- [PostgreSQL](#)
- [SQLite](#)

## ODBC

Allows ClickHouse to connect to external databases via [ODBC](#).

To safely implement ODBC connections, ClickHouse uses a separate program `clickhouse-odbc-bridge`. If the ODBC driver is loaded directly from `clickhouse-server`, driver problems can crash the ClickHouse server. ClickHouse automatically starts `clickhouse-odbc-bridge` when it is required. The ODBC bridge program is installed from the same package as the `clickhouse-server`.

This engine supports the [Nullable](#) data type.

## Creating a Table

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

See a detailed description of the [CREATE TABLE](#) query.

The table structure can differ from the source table structure:

- Column names should be the same as in the source table, but you can use just some of these columns and in any order.
- Column types may differ from those in the source table. ClickHouse tries to [cast](#) values to the ClickHouse data types.
- The [external\\_table\\_functions\\_use\\_nulls](#) setting defines how to handle Nullable columns. Default value: 1. If 0, the table function does not make Nullable columns and inserts default values instead of nulls. This is also applicable for NULL values inside arrays.

### Engine Parameters

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

## Usage Example

### Retrieving data from the local MySQL installation via ODBC

This example is checked for Ubuntu Linux 18.04 and MySQL server 5.7.

Ensure that unixODBC and MySQL Connector are installed.

By default (if installed from packages), ClickHouse starts as user clickhouse. Thus, you need to create and configure this user in the MySQL server.

```
$ sudo mysql
```

```
mysql> CREATE USER 'clickhouse'@'localhost' IDENTIFIED BY 'clickhouse';
mysql> GRANT ALL PRIVILEGES ON *.* TO 'clickhouse'@'clickhouse' WITH GRANT OPTION;
```

Then configure the connection in /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
```

You can check the connection using the isql utility from the unixODBC installation.

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

Table in MySQL:

```
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)
```

Table in ClickHouse, retrieving data from the MySQL table:

```
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

## See Also

- [ODBC external dictionaries](#)
- [ODBC table function](#)

## JDBC

Allows ClickHouse to connect to external databases via [JDBC](#).

To implement the JDBC connection, ClickHouse uses the separate program [clickhouse-jdbc-bridge](#) that should run as a daemon.

This engine supports the [Nullable](#) data type.

## Creating a Table

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name
(
    columns list...
)
ENGINE = JDBC(datasource_uri, external_database, external_table)
```

### Engine Parameters

- `datasource_uri` — URI or name of an external DBMS.

URI Format: `jdbc:<driver_name>://<host_name>:<port>/?user=<username>&password=<password>`.

Example for MySQL: `jdbc:mysql://localhost:3306/?user=root&password=root`.

- `external_database` — Database in an external DBMS.
- `external_table` — Name of the table in `external_database` or a select query like `select * from table1 where column1=1`.

## Usage Example

Creating a table in MySQL server by connecting directly with its console client:

```
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)
```

Creating a table in ClickHouse server and selecting data from it:

```
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

```
INSERT INTO jdbc_table(`int_id`, `float`)
SELECT toInt32(number), toFloat32(number * 1.0)
FROM system.numbers
```

## See Also

- [JDBC table function](#).

## MySQL

The MySQL engine allows you to perform SELECT and INSERT queries on data that is stored on a remote MySQL server.

## Creating a Table

```

CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
(
    name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1] [TTL expr1],
    name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2] [TTL expr2],
    ...
) ENGINE = MySQL('host:port', 'database', 'table', 'user', 'password'[, replace_query, 'on_duplicate_clause'])
SETTINGS
    [connection_pool_size=16, ]
    [connection_max_tries=3, ]
    [connection_wait_timeout=5, ] /* 0 -- do not wait */
    [connection_auto_close=true ]
;

```

See a detailed description of the [CREATE TABLE](#) query.

The table structure can differ from the original MySQL table structure:

- Column names should be the same as in the original MySQL table, but you can use just some of these columns and in any order.
- Column types may differ from those in the original MySQL table. ClickHouse tries to [cast](#) values to the ClickHouse data types.
- The [external\\_table\\_functions\\_use\\_nulls](#) setting defines how to handle Nullable columns. Default value: 1. If 0, the table function does not make Nullable columns and inserts default values instead of nulls. This is also applicable for NULL values inside arrays.

## Engine Parameters

- `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` queries to `REPLACE INTO`. If `replace_query=1`, the query is substituted.
- `on_duplicate_clause` — The `ON DUPLICATE KEY on_duplicate_clause` expression that is added to the `INSERT` query.

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.

Simple `WHERE` clauses such as `=, !=, >, >=, <, <=` are executed on the MySQL server.

The rest of the conditions and the `LIMIT` sampling constraint are executed in ClickHouse only after the query to MySQL finishes.

Supports multiple replicas that must be listed by `|`. For example:

```

CREATE TABLE test_replicas (id UInt32, name String, age UInt32, money UInt32) ENGINE =
MySQL(`mysql{2|3|4}:3306`, 'clickhouse', 'test_replicas', 'root', 'clickhouse');

```

# Usage Example

Table in MySQL:

```
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)

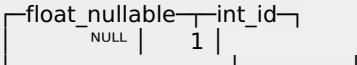
mysql> insert into test (`int_id`, `float`) VALUES (1,2);
Query OK, 1 row affected (0,00 sec)

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

Table in ClickHouse, retrieving data from the MySQL table created above:

```
CREATE TABLE mysql_table
(
  `float_nullable` Nullable(Float32),
  `int_id` Int32
)
ENGINE = MySQL('localhost:3306', 'test', 'test', 'bayonet', '123')
```

```
SELECT * FROM mysql_table
```



## See Also

- [The 'mysql' table function](#)
- [Using MySQL as a source of external dictionary](#)

## MongoDB

MongoDB engine is read-only table engine which allows to read data (SELECT queries) from remote MongoDB collection. Engine supports only non-nested data types. INSERT queries are not supported.

## Creating a Table

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name
(
  name1 [type1],
  name2 [type2],
  ...
) ENGINE = MongoDB(host:port, database, collection, user, password [, options]);
```

## Engine Parameters

- `host:port` — MongoDB server address.
- `database` — Remote database name.
- `collection` — Remote collection name.
- `user` — MongoDB user.
- `password` — User password.
- `options` — MongoDB connection string options (optional parameter).

## Usage Example

Create a table in ClickHouse which allows to read data from MongoDB collection:

```
CREATE TABLE mongo_table
(
    key UInt64,
    data String
) ENGINE = MongoDB('mongo1:27017', 'test', 'simple_table', 'testuser', 'clickhouse');
```

To read from an SSL secured MongoDB server:

```
CREATE TABLE mongo_table_ssl
(
    key UInt64,
    data String
) ENGINE = MongoDB('mongo2:27017', 'test', 'simple_table', 'testuser', 'clickhouse', 'ssl=true');
```

Query:

```
SELECT COUNT() FROM mongo_table;
```

count()
4

## HDFS

This engine provides integration with [Apache Hadoop](#) ecosystem by allowing to manage data on [HDFS](#) via ClickHouse. This engine is similar to the [File](#) and [URL](#) engines, but provides Hadoop-specific features.

## Usage

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

The `URI` parameter is the whole file URI in HDFS.

The `format` parameter specifies one of the available file formats. To perform `SELECT` queries, the format must be supported for input, and to perform `INSERT` queries – for output. The available formats are listed in the [Formats](#) section.

The path part of `URI` may contain globs. In this case the table would be readonly.

### Example:

## 1. Set up the `hdfs_engine_table` table:

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

## 2. Fill file:

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

## 3. Query the data:

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

name	value
one	1
two	2

# Implementation Details

- Reads and writes can be parallel.
- **Zero-copy** replication is supported.
- Not supported:
  - `ALTER` and `SELECT...SAMPLE` operations.
  - Indexes.

## Globs in path

Multiple path components can have globs. For being processed file should exists and matches to the whole path pattern. Listing of files determines during `SELECT` (not at `CREATE` moment).

- `*` — Substitutes any number of any characters except/ including empty string.
- `?` — 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.

Constructions with `{}` are similar to the **remote** table function.

## Example

1. Suppose we have several files in TSV format with the following URIs on HDFS:

- `'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. There are several ways to make a table consisting of all six files:

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

Another way:

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

Table consists of all the files in both directories (all files should satisfy format and schema described in query):

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

## Warning

If the listing of files contains number ranges with leading zeros, use the construction with braces for each digit separately or use `?`.

## Example

Create table with files named `file000`, `file001`, ... , `file999`:

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

## Configuration

Similar to GraphiteMergeTree, the HDFS engine supports extended configuration using the ClickHouse config file. There are two configuration keys that you can use: global (`hdfs`) and user-level (`hdfs_*`). The global configuration is applied first, and then the user-level configuration is applied (if it exists).

```
<!-- Global configuration options for HDFS engine type -->  
<hdfs>  
  <hadoop_kerberos_keytab>/tmp/keytab/clickhouse.keytab</hadoop_kerberos_keytab>  
  <hadoop_kerberos_principal>clickuser@TEST.CLICKHOUSE.TECH</hadoop_kerberos_principal>  
  <hadoop_security_authentication>kerberos</hadoop_security_authentication>  
</hdfs>  
  
<!-- Configuration specific for user "root" -->  
<hdfs_root>  
  <hadoop_kerberos_principal>root@TEST.CLICKHOUSE.TECH</hadoop_kerberos_principal>  
</hdfs_root>
```

## Configuration Options

Supported by libhdfs3

parameter	default value
rpc_client_connect_tcpnodelay	true
dfs_client_read_shortcircuit	true
output_replace-datanode-on-failure	true
input_notretry-another-node	false
input_localread_mappedfile	true
dfs_client_use_legacy_blockreader_local	false
rpc_client_ping_interval	10 * 1000
rpc_client_connect_timeout	600 * 1000
rpc_client_read_timeout	3600 * 1000
rpc_client_write_timeout	3600 * 1000
rpc_client_socekt_linger_timeout	-1
rpc_client_connect_retry	10
rpc_client_timeout	3600 * 1000
dfs_default_replica	3
input_connect_timeout	600 * 1000
input_read_timeout	3600 * 1000
input_write_timeout	3600 * 1000
input_localread_default_buffersize	1 * 1024 * 1024
dfs_prefetchsize	10
input_read_getblockinfo_retry	3
input_localread_blockinfo_cachesize	1000
input_read_max_retry	60
output_default_chunksize	512
output_default_packetsize	64 * 1024
output_default_write_retry	10
output_connect_timeout	600 * 1000
output_read_timeout	3600 * 1000
output_write_timeout	3600 * 1000
output_close_timeout	3600 * 1000
output_packetpool_size	1024
output_heeartbeat_interval	10 * 1000
dfs_client_failover_max_attempts	15
dfs_client_read_shortcircuit_streams_cache_size	256
dfs_client_socketcache_expiryMsec	3000
dfs_client_socketcache_capacity	16
dfs_default_blocksize	64 * 1024 * 1024
dfs_default_uri	"hdfs://localhost:9000"
hadoop_security_authentication	"simple"
hadoop_security_kerberos_ticket_cache_path	""
dfs_client_log_severity	"INFO"
dfs_domain_socket_path	""

[HDFS Configuration Reference](#) might explain some parameters.

## ClickHouse extras

parameter	default value
hadoop_kerberos_keytab	""
hadoop_kerberos_principal	""
hadoop_kerberos_kinit_command	kinit
libhdfs3_conf	""

## Limitations

- hadoop\_security\_kerberos\_ticket\_cache\_path and libhdfs3\_conf can be global only, not user specific

# Kerberos support

If `hadoop_security_authentication` parameter has value 'kerberos', ClickHouse authentifies via Kerberos facility.

Parameters [here](#) and `hadoop_security_kerberos_ticket_cache_path` may be of help.

Note that due to libhdfs3 limitations only old-fashioned approach is supported, datanode communications are not secured by SASL (`HADOOP_SECURE_DN_USER` is a reliable indicator of such security approach).

Use `tests/integration/test_storage_kerberized_hdfs/hdfs_configs/bootstrap.sh` for reference.

If `hadoop_kerberos_keytab`, `hadoop_kerberos_principal` or `hadoop_kerberos_kinit_command` is specified, kinit will be invoked. `hadoop_kerberos_keytab` and `hadoop_kerberos_principal` are mandatory in this case. kinit tool and krb5 configuration files are required.

## HDFS Namenode HA support{#namenode-ha}

libhdfs3 support HDFS namenode HA.

- Copy `hdfs-site.xml` from an HDFS node to `/etc/clickhouse-server/`.
- Add following piece to ClickHouse config file:

```
<hdfs>
  <libhdfs3_conf>/etc/clickhouse-server/hdfs-site.xml</libhdfs3_conf>
</hdfs>
```

- Then use `dfs.nameservices` tag value of `hdfs-site.xml` as the namenode address in the HDFS URI. For example, replace `hdfs://appadmin@192.168.101.11:8020/abc/` with `hdfs://appadmin@my_nameservice/abc/`.

## Virtual Columns

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

### See Also

- [Virtual columns](#)

## S3 Table Engine

This engine provides integration with [Amazon S3](#) ecosystem. This engine is similar to the [HDFS](#) engine, but provides S3-specific features.

## Create Table

```
CREATE TABLE s3_engine_table (name String, value UInt32)
ENGINE = S3(path, [aws_access_key_id, aws_secret_access_key,] format, [compression])
```

### Engine parameters

- `path` — Bucket url with path to file. Supports following wildcards in readonly mode: `*`, `?`, `{abc,def}` and `{N..M}` where `N, M` — numbers, `'abc'`, `'def'` — strings. For more information see [below](#).
- `format` — The [format](#) of the file.

- `aws_access_key_id`, `aws_secret_access_key` - Long-term credentials for the AWS account user. You can use these to authenticate your requests. Parameter is optional. If credentials are not specified, they are used from the configuration file. For more information see [Using S3 for Data Storage](#).
- `compression` — Compression type. Supported values: `none`, `gzip/gz`, `brotli/br`, `xz/LZMA`, `zstd/zst`. Parameter is optional. By default, it will autodetect compression by file extension.

## Example

1. Set up the `s3_engine_table` table:

```
CREATE TABLE s3_engine_table (name String, value UInt32) ENGINE=S3('https://storage.yandexcloud.net/my-test-bucket-768/test-data.csv.gz', 'CSV', 'gzip');
```

2. Fill file:

```
INSERT INTO s3_engine_table VALUES ('one', 1), ('two', 2), ('three', 3);
```

3. Query the data:

```
SELECT * FROM s3_engine_table LIMIT 2;
```

name	value
one	1
two	2

## Virtual columns

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

For more information about virtual columns see [here](#).

## Implementation Details

- Reads and writes can be parallel
- [Zero-copy](#) replication is supported.
- Not supported:
  - `ALTER` and `SELECT...SAMPLE` operations.
  - Indexes.

## Wildcards In Path

`path` argument can specify multiple files using bash-like wildcards. For being processed file should exist and match to the whole path pattern. Listing of files is determined during `SELECT` (not at `CREATE` moment).

- `*` — Substitutes any number of any characters except/ including empty string.
- `?` — 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. N and M can have leading zeroes e.g. `000..078`.

Constructions with `{}` are similar to the `remote` table function.

## Example

1. Suppose we have several files in CSV format with the following URIs on S3:

- `'https://storage.yandexcloud.net/my-test-bucket-768/some_prefix/some_file_1.csv'`
- `'https://storage.yandexcloud.net/my-test-bucket-768/some_prefix/some_file_2.csv'`
- `'https://storage.yandexcloud.net/my-test-bucket-768/some_prefix/some_file_3.csv'`
- `'https://storage.yandexcloud.net/my-test-bucket-768/another_prefix/some_file_1.csv'`
- `'https://storage.yandexcloud.net/my-test-bucket-768/another_prefix/some_file_2.csv'`
- `'https://storage.yandexcloud.net/my-test-bucket-768/another_prefix/some_file_3.csv'`

There are several ways to make a table consisting of all six files:

The first way:

```
CREATE TABLE table_with_range (name String, value UInt32) ENGINE = S3('https://storage.yandexcloud.net/my-test-bucket-768/{some,another}_prefix/some_file_{1..3}', 'CSV');
```

Another way:

```
CREATE TABLE table_with_question_mark (name String, value UInt32) ENGINE = S3('https://storage.yandexcloud.net/my-test-bucket-768/{some,another}_prefix/some_file_?', 'CSV');
```

Table consists of all the files in both directories (all files should satisfy format and schema described in query):

```
CREATE TABLE table_with_asterisk (name String, value UInt32) ENGINE = S3('https://storage.yandexcloud.net/my-test-bucket-768/{some,another}_prefix/*', 'CSV');
```

If the listing of files contains number ranges with leading zeros, use the construction with braces for each digit separately or use `?`.

## Example

Create table with files named `file-000.csv`, `file-001.csv`, ..., `file-999.csv`:

```
CREATE TABLE big_table (name String, value UInt32) ENGINE = S3('https://storage.yandexcloud.net/my-test-bucket-768/big_prefix/file-{000..999}.csv', 'CSV');
```

## Virtual Columns

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

## See Also

- [Virtual columns](#)

# S3-related settings

The following settings can be set before query execution or placed into configuration file.

- `s3_max_single_part_upload_size` — The maximum size of object to upload using singlepart upload to S3. Default value is `64Mb`.
- `s3_min_upload_part_size` — The minimum size of part to upload during multipart upload to [S3 Multipart upload](#). Default value is `512Mb`.
- `s3_max_redirects` — Max number of S3 redirects hops allowed. Default value is `10`.
- `s3_single_read_retries` — The maximum number of attempts during single read. Default value is `4`.

Security consideration: if malicious user can specify arbitrary S3 URLs, `s3_max_redirects` must be set to zero to avoid [SSRF](#) attacks; or alternatively, `remote_host_filter` must be specified in server configuration.

# Endpoint-based Settings

The following settings can be specified in configuration file for given endpoint (which will be matched by exact prefix of a URL):

- `endpoint` — Specifies prefix of an endpoint. Mandatory.
- `access_key_id` and `secret_access_key` — Specifies credentials to use with given endpoint. Optional.
- `use_environment_credentials` — If set to `true`, S3 client will try to obtain credentials from environment variables and [Amazon EC2](#) metadata for given endpoint. Optional, default value is `false`.
- `region` — Specifies S3 region name. Optional.
- `use_insecure_imds_request` — If set to `true`, S3 client will use insecure IMDS request while obtaining credentials from Amazon EC2 metadata. Optional, default value is `false`.
- `header` — Adds specified HTTP header to a request to given endpoint. Optional, can be specified multiple times.
- `server_side_encryption_customer_key_base64` — If specified, required headers for accessing S3 objects with SSE-C encryption will be set. Optional.
- `max_single_read_retries` — The maximum number of attempts during single read. Default value is `4`. Optional.

## Example:

```
<s3>
  <endpoint-name>
    <endpoint>https://storage.yandexcloud.net/my-test-bucket-768/</endpoint>
    <!-- <access_key_id>ACCESS_KEY_ID</access_key_id> -->
    <!-- <secret_access_key>SECRET_ACCESS_KEY</secret_access_key> -->
    <!-- <region>us-west-1</region> -->
    <!-- <use_environment_credentials>false</use_environment_credentials> -->
    <!-- <use_insecure_imds_request>false</use_insecure_imds_request> -->
    <!-- <header>Authorization: Bearer SOME-TOKEN</header> -->
    <!-- <server_side_encryption_customer_key_base64>BASE64-ENCODED-
KEY</server_side_encryption_customer_key_base64> -->
    <!-- <max_single_read_retries>4</max_single_read_retries> -->
  </endpoint-name>
</s3>
```

# Usage

Suppose we have several files in CSV format with the following URLs on S3:

- '[https://storage.yandexcloud.net/my-test-bucket-768/some\\_prefix/some\\_file\\_1.csv](https://storage.yandexcloud.net/my-test-bucket-768/some_prefix/some_file_1.csv)'
- '[https://storage.yandexcloud.net/my-test-bucket-768/some\\_prefix/some\\_file\\_2.csv](https://storage.yandexcloud.net/my-test-bucket-768/some_prefix/some_file_2.csv)'
- '[https://storage.yandexcloud.net/my-test-bucket-768/some\\_prefix/some\\_file\\_3.csv](https://storage.yandexcloud.net/my-test-bucket-768/some_prefix/some_file_3.csv)'
- '[https://storage.yandexcloud.net/my-test-bucket-768/another\\_prefix/some\\_file\\_1.csv](https://storage.yandexcloud.net/my-test-bucket-768/another_prefix/some_file_1.csv)'
- '[https://storage.yandexcloud.net/my-test-bucket-768/another\\_prefix/some\\_file\\_2.csv](https://storage.yandexcloud.net/my-test-bucket-768/another_prefix/some_file_2.csv)'
- '[https://storage.yandexcloud.net/my-test-bucket-768/another\\_prefix/some\\_file\\_3.csv](https://storage.yandexcloud.net/my-test-bucket-768/another_prefix/some_file_3.csv)'

1. There are several ways to make a table consisting of all six files:

```
CREATE TABLE table_with_range (name String, value UInt32)
ENGINE = S3('https://storage.yandexcloud.net/my-test-bucket-768/{some,another}_prefix/some_file_{1..3}', 'CSV');
```

2. Another way:

```
CREATE TABLE table_with_question_mark (name String, value UInt32)
ENGINE = S3('https://storage.yandexcloud.net/my-test-bucket-768/{some,another}_prefix/some_file_?', 'CSV');
```

3. Table consists of all the files in both directories (all files should satisfy format and schema described in query):

```
CREATE TABLE table_with_asterisk (name String, value UInt32)
ENGINE = S3('https://storage.yandexcloud.net/my-test-bucket-768/{some,another}_prefix/*', 'CSV');
```

## Warning

If the listing of files contains number ranges with leading zeros, use the construction with braces for each digit separately or use ?.

4. Create table with files named file-000.csv, file-001.csv, ... , file-999.csv:

```
CREATE TABLE big_table (name String, value UInt32)
ENGINE = S3('https://storage.yandexcloud.net/my-test-bucket-768/big_prefix/file-{000..999}.csv', 'CSV');
```

## See also

- [s3 table function](#)

## SQLite

The engine allows to import and export data to SQLite and supports queries to SQLite tables directly from ClickHouse.

## Creating a Table

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name
(
    name1 [type1],
    name2 [type2], ...
) ENGINE = SQLite('db_path', 'table')
```

## Engine Parameters

- `db_path` — Path to SQLite file with a database.
- `table` — Name of a table in the SQLite database.

## Usage Example

Shows a query creating the SQLite table:

```
SHOW CREATE TABLE sqlite_db.table2;
```

```
CREATE TABLE SQLite.table2
(
    `col1` Nullable(Int32),
    `col2` Nullable(String)
)
ENGINE = SQLite('sqlite.db','table2');
```

Returns the data from the table:

```
SELECT * FROM sqlite_db.table2 ORDER BY col1;
```

col1	col2
1	text1
2	text2
3	text3

## See Also

- [SQLite engine](#)
- [sqlite table function](#)

## Kafka

This engine works with [Apache Kafka](#).

Kafka lets you:

- Publish or subscribe to data flows.
- Organize fault-tolerant storage.
- Process streams as they become available.

## Creating a Table

```

CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
(
    name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1],
    name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2],
    ...
) ENGINE = Kafka()
SETTINGS
    kafka_broker_list = 'host:port',
    kafka_topic_list = 'topic1,topic2,...',
    kafka_group_name = 'group_name',
    kafka_format = 'data_format'[,]
    [kafka_row_delimiter = 'delimiter_symbol',]
    [kafka_schema = ",",]
    [kafka_num_consumers = N,]
    [kafka_max_block_size = 0,]
    [kafka_skip_broken_messages = N,]
    [kafka_commit_every_batch = 0,]
    [kafka_thread_per_consumer = 0]

```

#### Required parameters:

- `kafka_broker_list` — A comma-separated list of brokers (for example, `localhost:9092`).
- `kafka_topic_list` — A list of Kafka topics.
- `kafka_group_name` — A group of Kafka consumers. Reading margins are tracked for each group separately. If you do not want messages to be duplicated in the cluster, use the same group name everywhere.
- `kafka_format` — Message format. Uses the same notation as the SQL `FORMAT` function, such as `JSONEachRow`. For more information, see the [Formats](#) section.

#### Optional parameters:

- `kafka_row_delimiter` — Delimiter character, which ends the message.
- `kafka_schema` — Parameter that must be used if the format requires a schema definition. For example, [Cap'n Proto](#) requires the path to the schema file and the name of the `rootschema.capnp:Message` object.
- `kafka_num_consumers` — The number of consumers per table. Default: `1`. Specify more consumers if the throughput of one consumer is insufficient. The total number of consumers should not exceed the number of partitions in the topic, since only one consumer can be assigned per partition.
- `kafka_max_block_size` — The maximum batch size (in messages) for poll (default: `max_block_size`).
- `kafka_skip_broken_messages` — Kafka message parser tolerance to schema-incompatible messages per block. Default: `0`. If `kafka_skip_broken_messages = N` then the engine skips `N` Kafka messages that cannot be parsed (a message equals a row of data).
- `kafka_commit_every_batch` — Commit every consumed and handled batch instead of a single commit after writing a whole block (default: `0`).
- `kafka_thread_per_consumer` — Provide independent thread for each consumer (default: `0`). When enabled, every consumer flush the data independently, in parallel (otherwise — rows from several consumers squashed to form one block).

#### Examples:

```

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 queue3 (
    timestamp UInt64,
    level String,
    message String
) ENGINE = Kafka('localhost:9092', 'topic', 'group1')
    SETTINGS kafka_format = 'JSONEachRow',
    kafka_num_consumers = 4;

```

## ► Deprecated Method for Creating a Table

# Description

The delivered messages are tracked automatically, so each message in a group is only counted once. If you want to get the data twice, then create a copy of the table with another group name.

Groups are flexible and synced on the cluster. For instance, if you have 10 topics and 5 copies of a table in a cluster, then each copy gets 2 topics. If the number of copies changes, the topics are redistributed across the copies automatically. Read more about this at <http://kafka.apache.org/intro>.

`SELECT` is not particularly useful for reading messages (except for debugging), because each message can be read only once. It is more practical to create real-time threads using materialized views. To do this:

1. Use the engine to create a Kafka consumer and consider it a data stream.
2. Create a table with the desired structure.
3. Create a materialized view that converts data from the engine and puts it into a previously created table.

When the MATERIALIZED VIEW joins the engine, it starts collecting data in the background. This allows you to continually receive messages from Kafka and convert them to the required format using `SELECT`.

One kafka table can have as many materialized views as you like, they do not read data from the kafka table directly, but receive new records (in blocks), this way you can write to several tables with different detail level (with grouping - aggregation and without).

Example:

```

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;

```

To improve performance, received messages are grouped into blocks the size of `max_insert_block_size`. If the block wasn't formed within `stream_flush_interval_ms` milliseconds, the data will be flushed to the table regardless of the completeness of the block.

To stop receiving topic data or to change the conversion logic, detach the materialized view:

```

DETACH TABLE consumer;
ATTACH TABLE consumer;

```

If you want to change the target table by using `ALTER`, we recommend disabling the material view to avoid discrepancies between the target table and the data from the view.

## Configuration

Similar to GraphiteMergeTree, the Kafka engine supports extended configuration using the ClickHouse config file. There are two configuration keys that you can use: global (`kafka`) and topic-level (`kafka_*`). The global configuration is applied first, and then the topic-level configuration is applied (if it exists).

```

<!-- 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>

```

For a list of possible configuration options, see the [librdkafka configuration reference](#). Use the underscore (`_`) instead of a dot in the ClickHouse configuration. For example, `check.crcs=true` will be `<check_crcs>true</check_crcs>`.

## Kerberos support

To deal with Kerberos-aware Kafka, add `security_protocol` child element with `sasl_plaintext` value. It is enough if Kerberos ticket-granting ticket is obtained and cached by OS facilities.

ClickHouse is able to maintain Kerberos credentials using a keytab file. Consider `sasl_kerberos_service_name`, `sasl_kerberos_keytab`, `sasl_kerberos_principal` and `sasl.kerberos.kinit.cmd` child elements.

Example:

```

<!-- Kerberos-aware Kafka -->
<kafka>
  <security_protocol>SASL_PLAINTEXT</security_protocol>
  <sasl_kerberos_keytab>/home/kafkauser/kafkauser.keytab</sasl_kerberos_keytab>
  <sasl_kerberos_principal>kafkauser/kafkahost@EXAMPLE.COM</sasl_kerberos_principal>
</kafka>

```

## Virtual Columns

- `_topic` — Kafka topic.
- `_key` — Key of the message.
- `_offset` — Offset of the message.
- `_timestamp` — Timestamp of the message.
- `_partition` — Partition of Kafka topic.

### See Also

- [Virtual columns](#)
- [background\\_schedule\\_pool\\_size](#)

## EmbeddedRocksDB Engine

This engine allows integrating ClickHouse with [rocksdb](#).

### Creating a Table

```

CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
(
    name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1],
    name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2],
    ...
) ENGINE = EmbeddedRocksDB PRIMARY KEY(primary_key_name)

```

Required parameters:

- `primary_key_name` – any column name in the column list.
- `primary key` must be specified, it supports only one column in the primary key. The primary key will be serialized in binary as a `rocksdb` key.
- columns other than the primary key will be serialized in binary as `rocksdb` value in corresponding order.
- queries with key `equals` or `in` filtering will be optimized to multi keys lookup from `rocksdb`.

Example:

```

CREATE TABLE test
(
    `key` String,
    `v1` UInt32,
    `v2` String,
    `v3` Float32,
)
ENGINE = EmbeddedRocksDB
PRIMARY KEY key

```

## Metrics

There is also system.rocksdb table, that expose rocksdb statistics:

```
SELECT
    name,
    value
FROM system.rocksdb

[{"name": "no.file.opens", "value": 1}, {"name": "number.block.decompressed", "value": 1}]
```

## Configuration

You can also change any [rocksdb options](#) using config:

```
<rocksdb>
  <options>
    <max_background_jobs>8</max_background_jobs>
  </options>
  <column_family_options>
    <num_levels>2</num_levels>
  </column_family_options>
  <tables>
    <table>
      <name>TABLE</name>
      <options>
        <max_background_jobs>8</max_background_jobs>
      </options>
      <column_family_options>
        <num_levels>2</num_levels>
      </column_family_options>
    </table>
  </tables>
</rocksdb>
```

## RabbitMQ Engine

This engine allows integrating ClickHouse with [RabbitMQ](#).

RabbitMQ lets you:

- Publish or subscribe to data flows.
- Process streams as they become available.

## Creating a Table

```

CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
(
    name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1],
    name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2],
    ...
) ENGINE = RabbitMQ SETTINGS
    rabbitmq_host_port = 'host:port' [or rabbitmq_address = 'amqp(s)::guest:guest@localhost/vhost'],
    rabbitmq_exchange_name = 'exchange_name',
    rabbitmq_format = 'data_format'[,]
    [rabbitmq_exchange_type = 'exchange_type',]
    [rabbitmq_routing_key_list = 'key1,key2,...',]
    [rabbitmq_secure = 0,]
    [rabbitmq_row_delimiter = 'delimiter_symbol',]
    [rabbitmq_schema = ","]
    [rabbitmq_num_consumers = N,]
    [rabbitmq_num_queues = N,]
    [rabbitmq_queue_base = 'queue',]
    [rabbitmq_deadletter_exchange = 'dl-exchange',]
    [rabbitmq_persistent = 0,]
    [rabbitmq_skip_broken_messages = N,]
    [rabbitmq_max_block_size = N,]
    [rabbitmq_flush_interval_ms = N]

```

#### Required parameters:

- `rabbitmq_host_port` – host:port (for example, `localhost:5672`).
- `rabbitmq_exchange_name` – RabbitMQ exchange name.
- `rabbitmq_format` – Message format. Uses the same notation as the SQL `FORMAT` function, such as `JSONEachRow`. For more information, see the [Formats](#) section.

#### Optional parameters:

- `rabbitmq_exchange_type` – The type of RabbitMQ exchange: `direct`, `fanout`, `topic`, `headers`, `consistent_hash`. Default: `fanout`.
- `rabbitmq_routing_key_list` – A comma-separated list of routing keys.
- `rabbitmq_rowDelimiter` – Delimiter character, which ends the message.
- `rabbitmq_schema` – Parameter that must be used if the format requires a schema definition. For example, [Cap'n Proto](#) requires the path to the schema file and the name of the `rootschema.capnp:Message` object.
- `rabbitmq_num_consumers` – The number of consumers per table. Default: `1`. Specify more consumers if the throughput of one consumer is insufficient.
- `rabbitmq_num_queues` – Total number of queues. Default: `1`. Increasing this number can significantly improve performance.
- `rabbitmq_queue_base` - Specify a hint for queue names. Use cases of this setting are described below.
- `rabbitmq_deadletter_exchange` - Specify name for a [dead letter exchange](#). You can create another table with this exchange name and collect messages in cases when they are republished to dead letter exchange. By default dead letter exchange is not specified.
- `rabbitmq_persistent` - If set to `1` (true), in insert query delivery mode will be set to `2` (marks messages as 'persistent'). Default: `0`.
- `rabbitmq_skip_broken_messages` – RabbitMQ message parser tolerance to schema-incompatible messages per block. Default: `0`. If `rabbitmq_skip_broken_messages = N` then the engine skips `N` RabbitMQ messages that cannot be parsed (a message equals a row of data).
- `rabbitmq_max_block_size`

- `rabbitmq_flush_interval_ms`

SSL connection:

Use either `rabbitmq_secure = 1` or `amqps` in connection address: `rabbitmq_address = 'amqps://guest:guest@localhost/vhost'`.

The default behaviour of the used library is not to check if the created TLS connection is sufficiently secure. Whether the certificate is expired, self-signed, missing or invalid: the connection is simply permitted. More strict checking of certificates can possibly be implemented in the future.

Also format settings can be added along with rabbitmq-related settings.

Example:

```
CREATE TABLE queue (
    key UInt64,
    value UInt64,
    date DateTime
) ENGINE = RabbitMQ SETTINGS rabbitmq_host_port = 'localhost:5672',
    rabbitmq_exchange_name = 'exchange1',
    rabbitmq_format = 'JSONEachRow',
    rabbitmq_num_consumers = 5,
    date_time_input_format = 'best_effort';
```

The RabbitMQ server configuration should be added using the ClickHouse config file.

Required configuration:

```
<rabbitmq>
    <username>root</username>
    <password>clickhouse</password>
</rabbitmq>
```

Additional configuration:

```
<rabbitmq>
    <vhost>clickhouse</vhost>
</rabbitmq>
```

## Description

`SELECT` is not particularly useful for reading messages (except for debugging), because each message can be read only once. It is more practical to create real-time threads using [materialized views](#). To do this:

1. Use the engine to create a RabbitMQ consumer and consider it a data stream.
2. Create a table with the desired structure.
3. Create a materialized view that converts data from the engine and puts it into a previously created table.

When the `MATERIALIZED VIEW` joins the engine, it starts collecting data in the background. This allows you to continually receive messages from RabbitMQ and convert them to the required format using `SELECT`. One RabbitMQ table can have as many materialized views as you like.

Data can be channeled based on `rabbitmq_exchange_type` and the specified `rabbitmq_routing_key_list`. There can be no more than one exchange per table. One exchange can be shared between multiple tables - it enables routing into multiple tables at the same time.

Exchange type options:

- **direct** - Routing is based on the exact matching of keys. Example table key list: `key1,key2,key3,key4,key5`, message key can equal any of them.
- **fanout** - Routing to all tables (where exchange name is the same) regardless of the keys.
- **topic** - Routing is based on patterns with dot-separated keys. Examples: `*.logs`, `records.*.*.2020`, `*.2018,*.2019,*.2020`.
- **headers** - Routing is based on `key=value` matches with a setting `x-match=all` or `x-match=any`. Example table key list: `x-match=all,format=logs,type=report,year=2020`.
- **consistent\_hash** - Data is evenly distributed between all bound tables (where the exchange name is the same). Note that this exchange type must be enabled with RabbitMQ plugin: `rabbitmq-plugins enable rabbitmq_consistent_hash_exchange`.

Setting `rabbitmq_queue_base` may be used for the following cases:

- to let different tables share queues, so that multiple consumers could be registered for the same queues, which makes a better performance. If using `rabbitmq_num_consumers` and/or `rabbitmq_num_queues` settings, the exact match of queues is achieved in case these parameters are the same.
- to be able to restore reading from certain durable queues when not all messages were successfully consumed. To resume consumption from one specific queue - set its name in `rabbitmq_queue_base` setting and do not specify `rabbitmq_num_consumers` and `rabbitmq_num_queues` (defaults to 1). To resume consumption from all queues, which were declared for a specific table - just specify the same settings: `rabbitmq_queue_base`, `rabbitmq_num_consumers`, `rabbitmq_num_queues`. By default, queue names will be unique to tables.
- to reuse queues as they are declared durable and not auto-deleted. (Can be deleted via any of RabbitMQ CLI tools.)

To improve performance, received messages are grouped into blocks the size of `max_insert_block_size`. If the block wasn't formed within `stream_flush_interval_ms` milliseconds, the data will be flushed to the table regardless of the completeness of the block.

If `rabbitmq_num_consumers` and/or `rabbitmq_num_queues` settings are specified along with `rabbitmq_exchange_type`, then:

- `rabbitmq-consistent-hash-exchange` plugin must be enabled.
- `message_id` property of the published messages must be specified (unique for each message/batch).

For insert query there is message metadata, which is added for each published message: `messageID` and `republished` flag (true, if published more than once) - can be accessed via message headers.

Do not use the same table for inserts and materialized views.

Example:

```

CREATE TABLE queue (
    key UInt64,
    value UInt64
) ENGINE = RabbitMQ SETTINGS rabbitmq_host_port = 'localhost:5672',
    rabbitmq_exchange_name = 'exchange1',
    rabbitmq_exchange_type = 'headers',
    rabbitmq_routing_key_list = 'format=logs,type=report,year=2020',
    rabbitmq_format = 'JSONEachRow',
    rabbitmq_num_consumers = 5;

CREATE TABLE daily (key UInt64, value UInt64)
ENGINE = MergeTree() ORDER BY key;

CREATE MATERIALIZED VIEW consumer TO daily
AS SELECT key, value FROM queue;

SELECT key, value FROM daily ORDER BY key;

```

## Virtual Columns

- `_exchange_name` - RabbitMQ exchange name.
- `_channel_id` - ChannelID, on which consumer, who received the message, was declared.
- `_delivery_tag` - DeliveryTag of the received message. Scoped per channel.
- `_redelivered` - redelivered flag of the message.
- `_message_id` - messageID of the received message; non-empty if was set, when message was published.
- `_timestamp` - timestamp of the received message; non-empty if was set, when message was published.

## PostgreSQL

The PostgreSQL engine allows to perform `SELECT` and `INSERT` queries on data that is stored on a remote PostgreSQL server.

## Creating a Table

```

CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
(
    name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1] [TTL expr1],
    name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2] [TTL expr2],
    ...
) ENGINE = PostgreSQL('host:port', 'database', 'table', 'user', 'password'[, `schema`]);

```

See a detailed description of the [CREATE TABLE](#) query.

The table structure can differ from the original PostgreSQL table structure:

- Column names should be the same as in the original PostgreSQL table, but you can use just some of these columns and in any order.
- Column types may differ from those in the original PostgreSQL table. ClickHouse tries to [cast](#) values to the ClickHouse data types.
- The [external\\_table\\_functions\\_use\\_nulls](#) setting defines how to handle Nullable columns. Default value: 1. If 0, the table function does not make Nullable columns and inserts default values instead of nulls. This is also applicable for NULL values inside arrays.

## Engine Parameters

- `host:port` — PostgreSQL server address.
- `database` — Remote database name.
- `table` — Remote table name.
- `user` — PostgreSQL user.
- `password` — User password.
- `schema` — Non-default table schema. Optional.
- `on conflict ...` — example: `ON CONFLICT DO NOTHING`. Optional. Note: adding this option will make insertion less efficient.

## Implementation Details

`SELECT` queries on PostgreSQL side run as `COPY (SELECT ...)` TO STDOUT inside read-only PostgreSQL transaction with commit after each `SELECT` query.

Simple `WHERE` clauses such as `=`, `!=`, `>`, `>=`, `<`, `<=`, and `IN` are executed on the PostgreSQL server.

All joins, aggregations, sorting, `IN [ array ]` conditions and the `LIMIT` sampling constraint are executed in ClickHouse only after the query to PostgreSQL finishes.

`INSERT` queries on PostgreSQL side run as `COPY "table_name" (field1, field2, ... fieldN) FROM STDIN` inside PostgreSQL transaction with auto-commit after each `INSERT` statement.

PostgreSQL `Array` types are converted into ClickHouse arrays.

### Note

Be careful - in PostgreSQL an array data, created like a `type_name[]`, may contain multi-dimensional arrays of different dimensions in different table rows in same column. But in ClickHouse it is only allowed to have multidimensional arrays of the same count of dimensions in all table rows in same column.

Supports multiple replicas that must be listed by `|`. For example:

```
CREATE TABLE test_replicas (id UInt32, name String) ENGINE = PostgreSQL(`postgres{2|3|4}:5432`, 'clickhouse', 'test_replicas', 'postgres', 'mysecretpassword');
```

Replicas priority for PostgreSQL dictionary source is supported. The bigger the number in map, the less the priority. The highest priority is 0.

In the example below replica `example01-1` has the highest priority:

```

<postgresql>
  <port>5432</port>
  <user>clickhouse</user>
  <password>qwerty</password>
  <replica>
    <host>example01-1</host>
    <priority>1</priority>
  </replica>
  <replica>
    <host>example01-2</host>
    <priority>2</priority>
  </replica>
  <db>db_name</db>
  <table>table_name</table>
  <where>id=10</where>
  <invalidate_query>SQL_QUERY</invalidate_query>
</postgresql>
</source>

```

## Usage Example

Table in PostgreSQL:

```

postgres=# CREATE TABLE "public"."test" (
  "int_id" SERIAL,
  "int_nullable" INT NULL DEFAULT NULL,
  "float" FLOAT NOT NULL,
  "str" VARCHAR(100) NOT NULL DEFAULT '',
  "float_nullable" FLOAT NULL DEFAULT NULL,
  PRIMARY KEY (int_id));

CREATE TABLE

postgres=# INSERT INTO test (int_id, str, "float") VALUES (1,'test',2);
INSERT 0 1

postgres=# SELECT * FROM test;
 int_id | int_nullable | float | str | float_nullable
-----+-----+-----+-----+
  1 |           |    2 | test |
(1 row)

```

Table in ClickHouse, retrieving data from the PostgreSQL table created above:

```

CREATE TABLE default.postgresql_table
(
  `float_nullable` Nullable(Float32),
  `str` String,
  `int_id` Int32
)
ENGINE = PostgreSQL('localhost:5432', 'public', 'test', 'postges_user', 'postgres_password');

```

```

SELECT * FROM postgresql_table WHERE str IN ('test');

```

float_nullable	str	int_id
NULL	test	1

Using Non-default Schema:

```
postgres=# CREATE SCHEMA "nice.schema";
postgres=# CREATE TABLE "nice.schema"."nice.table" (a integer);
postgres=# INSERT INTO "nice.schema"."nice.table" SELECT i FROM generate_series(0, 99) as t(i)
```

```
CREATE TABLE pg_table_schema_with_dots (a UInt32)
    ENGINE PostgreSQL('localhost:5432', 'clickhouse', 'nice.table', 'postgrsql_user', 'password', 'nice.schema');
```

## See Also

- [The postgresql table function](#)
- [Using PostgreSQL as a source of external dictionary](#)

# ExternalDistributed

The ExternalDistributed engine allows to perform `SELECT` queries on data that is stored on a remote servers MySQL or PostgreSQL. Accepts [MySQL](#) or [PostgreSQL](#) engines as an argument so sharding is possible.

## Creating a Table

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
(
    name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1] [TTL expr1],
    name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2] [TTL expr2],
    ...
) ENGINE = ExternalDistributed('engine', 'host:port', 'database', 'table', 'user', 'password');
```

See a detailed description of the [CREATE TABLE](#) query.

The table structure can differ from the original table structure:

- Column names should be the same as in the original table, but you can use just some of these columns and in any order.
- Column types may differ from those in the original table. ClickHouse tries to [cast](#) values to the ClickHouse data types.

## Engine Parameters

- `engine` — The table engine MySQL or PostgreSQL.
- `host:port` — MySQL or PostgreSQL server address.
- `database` — Remote database name.
- `table` — Remote table name.
- `user` — User name.
- `password` — User password.

## Implementation Details

Supports multiple replicas that must be listed by | and shards must be listed by ,. For example:

```
CREATE TABLE test_shards (id UInt32, name String, age UInt32, money UInt32) ENGINE = ExternalDistributed('MySQL', `mysql{1|2}:3306,mysql{3|4}:3306`, 'clickhouse', 'test_replicas', 'root', 'clickhouse');
```

When specifying replicas, one of the available replicas is selected for each of the shards when reading. If the connection fails, the next replica is selected, and so on for all the replicas. If the connection attempt fails for all the replicas, the attempt is repeated the same way several times.

You can specify any number of shards and any number of replicas for each shard.

## See Also

- [MySQL table engine](#)
- [PostgreSQL table engine](#)
- [Distributed table engine](#)

## MaterializedPostgreSQL

Creates ClickHouse table with an initial data dump of PostgreSQL table and starts replication process, i.e. executes background job to apply new changes as they happen on PostgreSQL table in the remote PostgreSQL database.

If more than one table is required, it is highly recommended to use the [MaterializedPostgreSQL](#) database engine instead of the table engine and use the [materialized\\_postgresql\\_tables\\_list](#) setting, which specifies the tables to be replicated. It will be much better in terms of CPU, fewer connections and fewer replication slots inside the remote PostgreSQL database.

## Creating a Table

```
CREATE TABLE postgresql_db.postgresql_replica (key UInt64, value UInt64)
ENGINE = MaterializedPostgreSQL('postgres1:5432', 'postgres_database', 'postgresql_replica', 'postgres_user',
'postgres_password')
PRIMARY KEY key;
```

### Engine Parameters

- `host:port` — PostgreSQL server address.
- `database` — Remote database name.
- `table` — Remote table name.
- `user` — PostgreSQL user.
- `password` — User password.

## Requirements

1. The `wal_level` setting must have a value `logical` and `max_replication_slots` parameter must have a value at least 2 in the PostgreSQL config file.
2. A table with `MaterializedPostgreSQL` engine must have a primary key — the same as a replica identity index (by default: primary key) of a PostgreSQL table (see [details on replica identity index](#)).
3. Only database `Atomic` is allowed.

## Virtual columns

- `_version` — Transaction counter. Type: [UInt64](#).
- `_sign` — Deletion mark. Type: [Int8](#). Possible values:
  - `1` — Row is not deleted,
  - `-1` — Row is deleted.

These columns do not need to be added when a table is created. They are always accessible in `SELECT` query.

`_version` column equals `LSN` position in `WAL`, so it might be used to check how up-to-date replication is.

```
CREATE TABLE postgresql_db.postgresql_replica (key UInt64, value UInt64)
ENGINE = MaterializedPostgreSQL('postgres1:5432', 'postgres_database', 'postgresql_replica', 'postgres_user',
'postgres_password')
PRIMARY KEY key;

SELECT key, value, _version FROM postgresql_db.postgresql_replica;
```

## Warning

Replication of [TOAST](#) values is not supported. The default value for the data type will be used.

# Table Engines

The table engine (type of table) determines:

- How and where data is stored, where to write it to, and where to read it from.
- Which queries are supported, and how.
- Concurrent data access.
- Use of indexes, if present.
- Whether multithreaded request execution is possible.
- Data replication parameters.

## Engine Families

### MergeTree

The most universal and functional table engines for high-load tasks. The property shared by these engines is quick data insertion with subsequent background data processing. MergeTree family engines support data replication (with [Replicated\\*](#) versions of engines), partitioning, secondary data-skipping indexes, and other features not supported in other engines.

Engines in the family:

- [MergeTree](#)
- [ReplacingMergeTree](#)
- [SummingMergeTree](#)
- [AggregatingMergeTree](#)

- [CollapsingMergeTree](#)
- [VersionedCollapsingMergeTree](#)
- [GraphiteMergeTree](#)

## Log

Lightweight **engines** with minimum functionality. They're the most effective when you need to quickly write many small tables (up to approximately 1 million rows) and read them later as a whole.

Engines in the family:

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

## Integration Engines

Engines for communicating with other data storage and processing systems.

Engines in the family:

- [ODBC](#)
- [JDBC](#)
- [MySQL](#)
- [MongoDB](#)
- [HDFS](#)
- [S3](#)
- [Kafka](#)
- [EmbeddedRocksDB](#)
- [RabbitMQ](#)
- [PostgreSQL](#)

## Special Engines

Engines in the family:

- [Distributed](#)
- [MaterializedView](#)
- [Dictionary](#)
- [Merge](#)
- [File](#)
- [Null](#)
- [Set](#)
- [Join](#)

- [URL](#)
- [View](#)
- [Memory](#)
- [Buffer](#)

## Virtual Columns

Virtual column is an integral table engine attribute that is defined in the engine source code.

You shouldn't specify virtual columns in the `CREATE TABLE` query and you can't see them in `SHOW CREATE TABLE` and `DESCRIBE TABLE` query results. Virtual columns are also read-only, so you can't insert data into virtual columns.

To select data from a virtual column, you must specify its name in the `SELECT` query. `SELECT *` does not return values from virtual columns.

If you create a table with a column that has the same name as one of the table virtual columns, the virtual column becomes inaccessible. We do not recommend doing this. To help avoid conflicts, virtual column names are usually prefixed with an underscore.

---

## MergeTree Engine Family

Table engines from the MergeTree family are the core of ClickHouse data storage capabilities. They provide most features for resilience and high-performance data retrieval: columnar storage, custom partitioning, sparse primary index, secondary data-skipping indexes, etc.

Base [MergeTree](#) table engine can be considered the default table engine for single-node ClickHouse instances because it is versatile and practical for a wide range of use cases.

For production usage [ReplicatedMergeTree](#) is the way to go, because it adds high-availability to all features of regular MergeTree engine. A bonus is automatic data deduplication on data ingestion, so the software can safely retry if there was some network issue during insert.

All other engines of MergeTree family add extra functionality for some specific use cases. Usually, it's implemented as additional data manipulation in background.

The main downside of MergeTree engines is that they are rather heavy-weight. So the typical pattern is to have not so many of them. If you need many small tables, for example for temporary data, consider [Log engine family](#).

---

## MergeTree

The [MergeTree](#) engine and other engines of this family (`*MergeTree`) are the most robust ClickHouse table engines.

Engines in the MergeTree family are designed for inserting a very large amount of data into a table. The data is quickly written to the table part by part, then rules are applied for merging the parts in the background. This method is much more efficient than continually rewriting the data in storage during insert.

Main features:

- Stores data sorted by primary key.

This allows you to create a small sparse index that helps find data faster.

- Partitions can be used if the [partitioning key](#) is specified.

ClickHouse supports certain operations with partitions that are more efficient than general operations on the same data with the same result. ClickHouse also automatically cuts off the partition data where the partitioning key is specified in the query.

- Data replication support.

The family of ReplicatedMergeTree tables provides data replication. For more information, see [Data replication](#).

- Data sampling support.

If necessary, you can set the data sampling method in the table.

## Info

The **Merge** engine does not belong to the [\\*MergeTree family](#).

## Creating a Table

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
(
    name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1] [TTL expr1],
    name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2] [TTL expr2],
    ...
    INDEX index_name1 expr1 TYPE type1(...) GRANULARITY value1,
    INDEX index_name2 expr2 TYPE type2(...) GRANULARITY value2,
    ...
    PROJECTION projection_name_1 (SELECT <COLUMN LIST EXPR> [GROUP BY] [ORDER BY]),
    PROJECTION projection_name_2 (SELECT <COLUMN LIST EXPR> [GROUP BY] [ORDER BY])
) ENGINE = MergeTree()
ORDER BY expr
[PARTITION BY expr]
[PRIMARY KEY expr]
[SAMPLE BY expr]
[TTL expr
    [DELETE|TO DISK 'xxx'|TO VOLUME 'xxx' [, ...]]
    [WHERE conditions]
    [GROUP BY key_expr [SET v1 = aggr_func(v1) [, v2 = aggr_func(v2) ...]]]]
[SETTINGS name=value, ...]
```

For a description of parameters, see the [CREATE query description](#).

## Query Clauses

- **ENGINE** — Name and parameters of the engine. `ENGINE = MergeTree()`. The `MergeTree` engine does not have parameters.
- **ORDER BY** — The sorting key.

A tuple of column names or arbitrary expressions. Example: `ORDER BY (CounterID, EventDate)`.

ClickHouse uses the sorting key as a primary key if the primary key is not defined obviously by the `PRIMARY KEY` clause.

Use the `ORDER BY tuple()` syntax, if you do not need sorting. See [Selecting the Primary Key](#).

- **PARTITION BY** — The [partitioning key](#). Optional. In most cases you don't need partition key, and in most other cases you don't need partition key more granular than by months. Partitioning does not speed up queries (in contrast to the ORDER BY expression). You should never use too granular partitioning. Don't partition your data by client identifiers or names (instead make client identifier or name the first column in the ORDER BY expression).

For partitioning by month, use the `toYYYYMM(date_column)` expression, where `date_column` is a column with a date of the type [Date](#). The partition names here have the "YYYYMM" format.

- **PRIMARY KEY** — The primary key if it [differs from the sorting key](#). Optional.

By default the primary key is the same as the sorting key (which is specified by the `ORDER BY` clause). Thus in most cases it is unnecessary to specify a separate PRIMARY KEY clause.

- **SAMPLE BY** — An expression for sampling. Optional.

If a sampling expression is used, the primary key must contain it. The result of a sampling expression must be an unsigned integer. Example: `SAMPLE BY intHash32(UserID) ORDER BY (CounterID, EventDate, intHash32(UserID))`.

- **TTL** — A list of rules specifying storage duration of rows and defining logic of automatic parts movement [between disks and volumes](#). Optional.

Expression must have one `Date` or `DateTime` column as a result. Example:

`TTL date + INTERVAL 1 DAY`

Type of the rule `DELETE|TO DISK 'xxx'|TO VOLUME 'xxx'|GROUP BY` specifies an action to be done with the part if the expression is satisfied (reaches current time): removal of expired rows, moving a part (if expression is satisfied for all rows in a part) to specified disk (`TO DISK 'xxx'`) or to volume (`TO VOLUME 'xxx'`), or aggregating values in expired rows. Default type of the rule is removal (`DELETE`). List of multiple rules can be specified, but there should be no more than one `DELETE` rule.

For more details, see [TTL for columns and tables](#)

- **SETTINGS** — Additional parameters that control the behavior of the `MergeTree` (optional):

- `index_granularity` — Maximum number of data rows between the marks of an index. Default value: 8192. See [Data Storage](#).
- `index_granularity_bytes` — Maximum size of data granules in bytes. Default value: 10Mb. To restrict the granule size only by number of rows, set to 0 (not recommended). See [Data Storage](#).
- `min_index_granularity_bytes` — Min allowed size of data granules in bytes. Default value: 1024b. To provide a safeguard against accidentally creating tables with very low `index_granularity_bytes`. See [Data Storage](#).
- `enable_mixed_granularity_parts` — Enables or disables transitioning to control the granule size with the `index_granularity_bytes` setting. Before version 19.11, there was only the `index_granularity` setting for restricting granule size. The `index_granularity_bytes` setting improves ClickHouse performance when selecting data from tables with big rows (tens and hundreds of megabytes). If you have tables with big rows, you can enable this setting for the tables to improve the efficiency of `SELECT` queries.
- `use_minimalistic_part_header_in_zookeeper` — Storage method of the data parts headers in ZooKeeper. If `use_minimalistic_part_header_in_zookeeper=1`, then ZooKeeper stores less data. For more information, see the [setting description](#) in "Server configuration parameters".

- `min_merge_bytes_to_use_direct_io` — The minimum data volume for merge operation that is required for using direct I/O access to the storage disk. When merging data parts, ClickHouse calculates the total storage volume of all the data to be merged. If the volume exceeds `min_merge_bytes_to_use_direct_io` bytes, ClickHouse reads and writes the data to the storage disk using the direct I/O interface (`O_DIRECT` option). If `min_merge_bytes_to_use_direct_io = 0`, then direct I/O is disabled. Default value: `10 * 1024 * 1024 * 1024` bytes.
- `merge_with_ttl_timeout` — Minimum delay in seconds before repeating a merge with delete TTL. Default value: `14400` seconds (4 hours).
- `merge_with_recompression_ttl_timeout` — Minimum delay in seconds before repeating a merge with recompression TTL. Default value: `14400` seconds (4 hours).
- `try_fetch_recompressed_part_timeout` — Timeout (in seconds) before starting merge with recompression. During this time ClickHouse tries to fetch recompressed part from replica which assigned this merge with recompression. Default value: `7200` seconds (2 hours).
- `write_final_mark` — Enables or disables writing the final index mark at the end of data part (after the last byte). Default value: 1. Don't turn it off.
- `merge_max_block_size` — Maximum number of rows in block for merge operations. Default value: `8192`.
- `storage_policy` — Storage policy. See [Using Multiple Block Devices for Data Storage](#).
- `min_bytes_for_wide_part`, `min_rows_for_wide_part` — Minimum number of bytes/rows in a data part that can be stored in Wide format. You can set one, both or none of these settings. See [Data Storage](#).
- `max_parts_in_total` — Maximum number of parts in all partitions.
- `max_compress_block_size` — Maximum size of blocks of uncompressed data before compressing for writing to a table. You can also specify this setting in the global settings (see [max\\_compress\\_block\\_size](#) setting). The value specified when table is created overrides the global value for this setting.
- `min_compress_block_size` — Minimum size of blocks of uncompressed data required for compression when writing the next mark. You can also specify this setting in the global settings (see [min\\_compress\\_block\\_size](#) setting). The value specified when table is created overrides the global value for this setting.
- `max_partitions_to_read` — Limits the maximum number of partitions that can be accessed in one query. You can also specify setting [max\\_partitions\\_to\\_read](#) in the global setting.

## Example of Sections Setting

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

In the example, we set partitioning by month.

We also set an expression for sampling as a hash by the user ID. This allows you to pseudorandomize the data in the table for each `CounterID` and `EventDate`. If you define a `SAMPLE` clause when selecting the data, ClickHouse will return an evenly pseudorandom data sample for a subset of users.

The `index_granularity` setting can be omitted because `8192` is the default value.

### ► Deprecated Method for Creating a Table

## Data Storage

A table consists of data parts sorted by primary key.

When data is inserted in a table, separate data parts are created and each of them is lexicographically sorted by primary key. For example, if the primary key is `(CounterID, Date)`, the data in the part is sorted by `CounterID`, and within each `CounterID`, it is ordered by `Date`.

Data belonging to different partitions are separated into different parts. In the background, ClickHouse merges data parts for more efficient storage. Parts belonging to different partitions are not merged. The merge mechanism does not guarantee that all rows with the same primary key will be in the same data part.

Data parts can be stored in `Wide` or `Compact` format. In `Wide` format each column is stored in a separate file in a filesystem, in `Compact` format all columns are stored in one file. `Compact` format can be used to increase performance of small and frequent inserts.

Data storing format is controlled by the `min_bytes_for_wide_part` and `min_rows_for_wide_part` settings of the table engine. If the number of bytes or rows in a data part is less than the corresponding setting's value, the part is stored in `Compact` format. Otherwise it is stored in `Wide` format. If none of these settings is set, data parts are stored in `Wide` format.

Each data part is logically divided into granules. A granule is the smallest indivisible data set that ClickHouse reads when selecting data. ClickHouse does not split rows or values, so each granule always contains an integer number of rows. The first row of a granule is marked with the value of the primary key for the row. For each data part, ClickHouse creates an index file that stores the marks. For each column, whether it's in the primary key or not, ClickHouse also stores the same marks. These marks let you find data directly in column files.

The granule size is restricted by the `index_granularity` and `index_granularity_bytes` settings of the table engine. The number of rows in a granule lies in the  $[1, \text{index\_granularity}]$  range, depending on the size of the rows. The size of a granule can exceed `index_granularity_bytes` if the size of a single row is greater than the value of the setting. In this case, the size of the granule equals the size of the row.

## Primary Keys and Indexes in Queries

Take the `(CounterID, Date)` primary key as an example. In this case, the sorting and index can be illustrated as follows:

Whole data:	[-----]
CounterID:	[aaaaaaaaaaaaaaaaaaaaabbbbcdeeeeeeeeeeefgggggggghhhhhhhhiiiiiiikl       ]
Date:	[11111112222222333123321111222223321111121222222311112223311122333]
Marks:	a,1   a,2   a,3   b,3   e,2   e,3   g,1   h,2   i,1   i,3   l,3
Marks numbers:	0    1    2    3    4    5    6    7    8    9    10

If the data query specifies:

- `CounterID in ('a', 'h')`, the server reads the data in the ranges of marks `[0, 3]` and `[6, 8]`.
- `CounterID IN ('a', 'h') AND Date = 3`, the server reads the data in the ranges of marks `[1, 3]` and `[7, 8]`.
- `Date = 3`, the server reads the data in the range of marks `[1, 10]`.

The examples above show that it is always more effective to use an index than a full scan.

A sparse index allows extra data to be read. When reading a single range of the primary key, up to `index_granularity * 2` extra rows in each data block can be read.

Sparse indexes allow you to work with a very large number of table rows, because in most cases, such indexes fit in the computer's RAM.

ClickHouse does not require a unique primary key. You can insert multiple rows with the same primary key.

You can use Nullable-typed expressions in the PRIMARY KEY and ORDER BY clauses but it is strongly discouraged. To allow this feature, turn on the `allow_nullable_key` setting. The **NULLS\_LAST** principle applies for `NULL` values in the ORDER BY clause.

## Selecting the Primary Key

The number of columns in the primary key is not explicitly limited. Depending on the data structure, you can include more or fewer columns in the primary key. This may:

- Improve the performance of an index.

If the primary key is `(a, b)`, then adding another column `c` will improve the performance if the following conditions are met:

- There are queries with a condition on column `c`.
- Long data ranges (several times longer than the `index_granularity`) with identical values for `(a, b)` are common. In other words, when adding another column allows you to skip quite long data ranges.

- Improve data compression.

ClickHouse sorts data by primary key, so the higher the consistency, the better the compression.

- Provide additional logic when merging data parts in the **CollapsingMergeTree** and **SummingMergeTree** engines.

In this case it makes sense to specify the *sorting key* that is different from the primary key.

A long primary key will negatively affect the insert performance and memory consumption, but extra columns in the primary key do not affect ClickHouse performance during `SELECT` queries.

You can create a table without a primary key using the `ORDER BY tuple()` syntax. In this case, ClickHouse stores data in the order of inserting. If you want to save data order when inserting data by `INSERT ... SELECT` queries, set `max_insert_threads = 1`.

To select data in the initial order, use `single-threaded` `SELECT` queries.

## Choosing a Primary Key that Differs from the Sorting Key

It is possible to specify a primary key (an expression with values that are written in the index file for each mark) that is different from the sorting key (an expression for sorting the rows in data parts). In this case the primary key expression tuple must be a prefix of the sorting key expression tuple.

This feature is helpful when using the **SummingMergeTree** and **AggregatingMergeTree** table engines. In a common case when using these engines, the table has two types of columns: *dimensions* and *measures*. Typical queries aggregate values of measure columns with arbitrary GROUP BY and filtering by dimensions. Because SummingMergeTree and AggregatingMergeTree aggregate rows with the same value of the sorting key, it is natural to add all dimensions to it. As a result, the key expression consists of a long list of columns and this list must be frequently updated with newly added dimensions.

In this case it makes sense to leave only a few columns in the primary key that will provide efficient range scans and add the remaining dimension columns to the sorting key tuple.

**ALTER** of the sorting key is a lightweight operation because when a new column is simultaneously added to the table and to the sorting key, existing data parts do not need to be changed. Since the old sorting key is a prefix of the new sorting key and there is no data in the newly added column, the data is sorted by both the old and new sorting keys at the moment of table modification.

## Use of Indexes and Partitions in Queries

For `SELECT` queries, ClickHouse analyzes whether an index can be used. An index can be used if the `WHERE/PREWHERE` clause has an expression (as one of the conjunction elements, or entirely) that represents an equality or inequality comparison operation, or if it has `IN` or `LIKE` with a fixed prefix on columns or expressions that are in the primary key or partitioning key, or on certain partially repetitive functions of these columns, or logical relationships of these expressions.

Thus, it is possible to quickly run queries on one or many ranges of the primary key. In this example, queries will be fast when run for a specific tracking tag, for a specific tag and date range, for a specific tag and date, for multiple tags with a date range, and so on.

Let's look at the engine configured as follows:

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

In this case, in queries:

```
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 will use the primary key index to trim improper data and the monthly partitioning key to trim partitions that are in improper date ranges.

The queries above show that the index is used even for complex expressions. Reading from the table is organized so that using the index can't be slower than a full scan.

In the example below, the index can't be used.

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

To check whether ClickHouse can use the index when running a query, use the settings `force_index_by_date` and `force_primary_key`.

The key for partitioning by month allows reading only those data blocks which contain dates from the proper range. In this case, the data block may contain data for many dates (up to an entire month). Within a block, data is sorted by primary key, which might not contain the date as the first column. Because of this, using a query with only a date condition that does not specify the primary key prefix will cause more data to be read than for a single date.

## Use of Index for Partially-monotonic Primary Keys

Consider, for example, the days of the month. They form a [monotonic sequence](#) for one month, but not monotonic for more extended periods. This is a partially-monotonic sequence. If a user creates the table with partially-monotonic primary key, ClickHouse creates a sparse index as usual. When a user selects data from this kind of table, ClickHouse analyzes the query conditions. If the user wants to get data

between two marks of the index and both these marks fall within one month, ClickHouse can use the index in this particular case because it can calculate the distance between the parameters of a query and index marks.

ClickHouse cannot use an index if the values of the primary key in the query parameter range do not represent a monotonic sequence. In this case, ClickHouse uses the full scan method.

ClickHouse uses this logic not only for days of the month sequences, but for any primary key that represents a partially-monotonic sequence.

## Data Skipping Indexes

The index declaration is in the columns section of the `CREATE` query.

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

For tables from the `*MergeTree` family, data skipping indices can be specified.

These indices aggregate some information about the specified expression on blocks, which consist of `granularity_value` granules (the size of the granule is specified using the `index_granularity` setting in the table engine). Then these aggregates are used in `SELECT` queries for reducing the amount of data to read from the disk by skipping big blocks of data where the `where` query cannot be satisfied.

### Example

```
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()
...
```

Indices from the example can be used by ClickHouse to reduce the amount of data to read from disk in the following queries:

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

### Available Types of Indices

- `minmax`

Stores extremes of the specified expression (if the expression is `tuple`, then it stores extremes for each element of `tuple`), uses stored info for skipping blocks of data like the primary key.

- `set(max_rows)`

Stores unique values of the specified expression (no more than `max_rows` rows, `max_rows=0` means “no limits”). Uses the values to check if the `WHERE` expression is not satisfiable on a block of data.

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

Stores a **Bloom filter** that contains all ngrams from a block of data. Works only with datatypes: **String**, **FixedString** and **Map**. Can be used for optimization of **EQUALS**, **LIKE** and **IN** expressions.

- `n` — ngram size,
- `size_of_bloom_filter_in_bytes` — Bloom filter size in bytes (you can use large values here, for example, 256 or 512, because it can be compressed well).
- `number_of_hash_functions` — The number of hash functions used in the Bloom filter.
- `random_seed` — The seed for Bloom filter hash functions.

- `tokenbf_v1(size_of_bloom_filter_in_bytes, number_of_hash_functions, random_seed)`

The same as `ngrambf_v1`, but stores tokens instead of ngrams. Tokens are sequences separated by non-alphanumeric characters.

- `bloom_filter([false_positive])` — Stores a **Bloom filter** for the specified columns.

The optional `false_positive` parameter is the probability of receiving a false positive response from the filter. Possible values: (0, 1). Default value: 0.025.

Supported data types: `Int*`, `UInt*`, `Float*`, `Enum`, `Date`, `DateTime`, `String`, `FixedString`, `Array`, `LowCardinality`, `Nullable`, `UUID`, `Map`.

For `Map` data type client can specify if index should be created for keys or values using `mapKeys` or `mapValues` function.

The following functions can use the filter: `equals`, `notEquals`, `in`, `notin`, `has`.

Example of index creation for `Map` data type

```
INDEX map_key_index mapKeys(map_column) TYPE bloom_filter GRANULARITY 1
INDEX map_key_index mapValues(map_column) TYPE bloom_filter GRANULARITY 1
```

```
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
```

## Functions Support

Conditions in the `WHERE` clause contains calls of the functions that operate with columns. If the column is a part of an index, ClickHouse tries to use this index when performing the functions. ClickHouse supports different subsets of functions for using indexes.

The `set` index can be used with all functions. Function subsets for other indexes are shown in the table below.

Function (operator) / Index	primary key	minmax	ngrambf_v1	tokenbf_v1	bloom_filter
<code>equals (=, ==)</code>	✓	✓	✓	✓	✓
<code>notEquals(!=, &lt;&gt;)</code>	✓	✓	✓	✓	✓
<code>like</code>	✓	✓	✓	✓	✗

Function (operator) / Index	primary key	minmax	ngrambf_v1	tokenbf_v1	bloom_filter
notLike	✓	✓	✓	✓	✗
startsWith	✓	✓	✓	✓	✗
endsWith	✗	✗	✓	✓	✗
multiSearchAny	✗	✗	✓	✗	✗
in	✓	✓	✓	✓	✓
notIn	✓	✓	✓	✓	✓
less (<)	✓	✓	✗	✗	✗
greater (>)	✓	✓	✗	✗	✗
lessOrEquals (<=)	✓	✓	✗	✗	✗
greaterOrEquals (>=)	✓	✓	✗	✗	✗
empty	✓	✓	✗	✗	✗
notEmpty	✓	✓	✗	✗	✗
hasToken	✗	✗	✗	✓	✗

Functions with a constant argument that is less than ngram size can't be used by `ngrambf_v1` for query optimization.

## Note

Bloom filters can have false positive matches, so the `ngrambf_v1`, `tokenbf_v1`, and `bloom_filter` indexes can't be used for optimizing queries where the result of a function is expected to be false, for example:

- Can be optimized:
  - `s LIKE '%test%'`
  - `NOT s NOT LIKE '%test%'`
  - `s = 1`
  - `NOT s != 1`
  - `startsWith(s, 'test')`

- Can't be optimized:
  - NOT s LIKE '%test%'
  - s NOT LIKE '%test%'
  - NOT s = 1
  - s != 1
  - NOT startsWith(s, 'test')

## Projections

Projections are like [materialized views](#) but defined in part-level. It provides consistency guarantees along with automatic usage in queries.

Projections are an experimental feature. To enable them you must set the `allow_experimental_projection_optimization` to 1. See also the `force_optimize_projection` setting.

Projections are not supported in the `SELECT` statements with the `FINAL` modifier.

## Projection Query

A projection query is what defines a projection. It implicitly selects data from the parent table.

### Syntax

```
SELECT <column list expr> [GROUP BY] <group keys expr> [ORDER BY] <expr>
```

Projections can be modified or dropped with the [ALTER](#) statement.

## Projection Storage

Projections are stored inside the part directory. It's similar to an index but contains a subdirectory that stores an anonymous MergeTree table's part. The table is induced by the definition query of the projection. If there is a `GROUP BY` clause, the underlying storage engine becomes [AggregatingMergeTree](#), and all aggregate functions are converted to [AggregateFunction](#). If there is an `ORDER BY` clause, the `MergeTree` table uses it as its primary key expression. During the merge process the projection part is merged via its storage's merge routine. The checksum of the parent table's part is combined with the projection's part. Other maintenance jobs are similar to skip indices.

## Query Analysis

1. Check if the projection can be used to answer the given query, that is, it generates the same answer as querying the base table.
2. Select the best feasible match, which contains the least granules to read.
3. The query pipeline which uses projections will be different from the one that uses the original parts. If the projection is absent in some parts, we can add the pipeline to "project" it on the fly.

## Concurrent Data Access

For concurrent table access, we use multi-versioning. In other words, when a table is simultaneously read and updated, data is read from a set of parts that is current at the time of the query. There are no lengthy locks. Inserts do not get in the way of read operations.

Reading from a table is automatically parallelized.

## TTL for Columns and Tables

Determines the lifetime of values.

The `TTL` clause can be set for the whole table and for each individual column. Table-level `TTL` can also specify the logic of automatic moving data between disks and volumes, or recompressing parts where all the data has been expired.

Expressions must evaluate to `Date` or `DateTime` data type.

## Syntax

Setting time-to-live for a column:

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

To define interval, use `time interval` operators, for example:

```
TTL date_time + INTERVAL 1 MONTH  
TTL date_time + INTERVAL 15 HOUR
```

## Column TTL

When the values in the column expire, ClickHouse replaces them with the default values for the column data type. If all the column values in the data part expire, ClickHouse deletes this column from the data part in a filesystem.

The `TTL` clause can't be used for key columns.

## Examples

Creating a table with `TTL`:

```
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;
```

Adding `TTL` to a column of an existing table

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

Altering `TTL` of the column

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

## Table TTL

Table can have an expression for removal of expired rows, and multiple expressions for automatic move of parts between `disks or volumes`. When rows in the table expire, ClickHouse deletes all corresponding rows. For parts moving or recompressing, all rows of a part must satisfy the `TTL` expression criteria.

```

TTL expr
  [DELETE|RECOMPRESS codec_name1|TO DISK 'xxx'|TO VOLUME 'xxx'][, DELETE|RECOMPRESS codec_name2|TO
DISK 'aaa'|TO VOLUME 'bbb'] ...
  [WHERE conditions]
  [GROUP BY key_expr [SET v1 = aggr_func(v1) [, v2 = aggr_func(v2) ...]] ]

```

Type of TTL rule may follow each TTL expression. It affects an action which is to be done once the expression is satisfied (reaches current time):

- **DELETE** - delete expired rows (default action);
- **RECOMPRESS codec\_name** - recompress data part with the `codec_name`;
- **TO DISK 'aaa'** - move part to the disk `aaa`;
- **TO VOLUME 'bbb'** - move part to the disk `bbb`;
- **GROUP BY** - aggregate expired rows.

With `WHERE` clause you may specify which of the expired rows to delete or aggregate (it cannot be applied to moves or recompression).

`GROUP BY` expression must be a prefix of the table primary key.

If a column is not part of the `GROUP BY` expression and is not set explicitly in the `SET` clause, in result row it contains an occasional value from the grouped rows (as if aggregate function `any` is applied to it).

## Examples

Creating a table with TTL:

```

CREATE TABLE example_table
(
  d DateTime,
  a Int
)
ENGINE = MergeTree
PARTITION BY toYYYYMM(d)
ORDER BY d
TTL d + INTERVAL 1 MONTH [DELETE],
  d + INTERVAL 1 WEEK TO VOLUME 'aaa',
  d + INTERVAL 2 WEEK TO DISK 'bbb';

```

Altering TTL of the table:

```

ALTER TABLE example_table
  MODIFY TTL d + INTERVAL 1 DAY;

```

Creating a table, where the rows are expired after one month. The expired rows where dates are Mondays are deleted:

```

CREATE TABLE table_with_where
(
  d DateTime,
  a Int
)
ENGINE = MergeTree
PARTITION BY toYYYYMM(d)
ORDER BY d
TTL d + INTERVAL 1 MONTH DELETE WHERE toDayOfWeek(d) = 1;

```

Creating a table, where expired rows are recompressed:

```

CREATE TABLE table_for_recompression
(
    d DateTime,
    key UInt64,
    value String
) ENGINE MergeTree()
ORDER BY tuple()
PARTITION BY key
TTL d + INTERVAL 1 MONTH RECOMPRESS CODEC(ZSTD(17)), d + INTERVAL 1 YEAR RECOMPRESS CODEC(LZ4HC(10))
SETTINGS min_rows_for_wide_part = 0, min_bytes_for_wide_part = 0;

```

Creating a table, where expired rows are aggregated. In result rows `x` contains the maximum value accross the grouped rows, `y` — the minimum value, and `d` — any occasional value from grouped rows.

```

CREATE TABLE table_for_aggregation
(
    d DateTime,
    k1 Int,
    k2 Int,
    x Int,
    y Int
)
ENGINE = MergeTree
ORDER BY (k1, k2)
TTL d + INTERVAL 1 MONTH GROUP BY k1, k2 SET x = max(x), y = min(y);

```

## Removing Expired Data

Data with an expired `TTL` is removed when ClickHouse merges data parts.

When ClickHouse detects that data is expired, it performs an off-schedule merge. To control the frequency of such merges, you can set `merge_with_ttl_timeout`. If the value is too low, it will perform many off-schedule merges that may consume a lot of resources.

If you perform the `SELECT` query between merges, you may get expired data. To avoid it, use the `OPTIMIZE` query before `SELECT`.

### See Also

- [ttl\\_only\\_drop\\_parts](#) setting

## Using Multiple Block Devices for Data Storage

### Introduction

MergeTree family table engines can store data on multiple block devices. For example, it can be useful when the data of a certain table are implicitly split into “hot” and “cold”. The most recent data is regularly requested but requires only a small amount of space. On the contrary, the fat-tailed historical data is requested rarely. If several disks are available, the “hot” data may be located on fast disks (for example, NVMe SSDs or in memory), while the “cold” data - on relatively slow ones (for example, HDD).

Data part is the minimum movable unit for MergeTree-engine tables. The data belonging to one part are stored on one disk. Data parts can be moved between disks in the background (according to user settings) as well as by means of the `ALTER` queries.

### Terms

- Disk — Block device mounted to the filesystem.
- Default disk — Disk that stores the path specified in the `path` server setting.
- Volume — Ordered set of equal disks (similar to [JBOD](#)).

- Storage policy — Set of volumes and the rules for moving data between them.

The names given to the described entities can be found in the system tables, `system.storage_policies` and `system.disks`. To apply one of the configured storage policies for a table, use the `storage_policy` setting of MergeTree-engine family tables.

## Configuration

Disks, volumes and storage policies should be declared inside the `<storage_configuration>` tag either in the main file `config.xml` or in a distinct file in the `config.d` directory.

Configuration structure:

```
<storage_configuration>
  <disks>
    <disk_name_1> <!-- disk name -->
      <path>/mnt/fast_ssdclickhouse/</path>
    </disk_name_1>
    <disk_name_2>
      <path>/mnt/hdd1/clickhouse/</path>
      <keep_free_space_bytes>10485760</keep_free_space_bytes>
    </disk_name_2>
    <disk_name_3>
      <path>/mnt/hdd2/clickhouse/</path>
      <keep_free_space_bytes>10485760</keep_free_space_bytes>
    </disk_name_3>

    ...
  </disks>

  ...
</storage_configuration>
```

Tags:

- `<disk_name_N>` — Disk name. Names must be different for all disks.
- `path` — path under which a server will store data (`data` and `shadow` folders), should be terminated with '/'.
- `keep_free_space_bytes` — the amount of free disk space to be reserved.

The order of the disk definition is not important.

Storage policies configuration markup:

```

<storage_configuration>
...
<policies>
  <policy_name_1>
    <volumes>
      <volume_name_1>
        <disk>disk_name_from_disks_configuration</disk>
        <max_data_part_size_bytes>1073741824</max_data_part_size_bytes>
      </volume_name_1>
      <volume_name_2>
        <!-- configuration -->
      </volume_name_2>
      <!-- more volumes -->
    </volumes>
    <move_factor>0.2</move_factor>
  </policy_name_1>
  <policy_name_2>
    <!-- configuration -->
  </policy_name_2>

  <!-- more policies -->
</policies>
...
</storage_configuration>

```

Tags:

- `policy_name_N` — Policy name. Policy names must be unique.
- `volume_name_N` — Volume name. Volume names must be unique.
- `disk` — a disk within a volume.
- `max_data_part_size_bytes` — the maximum size of a part that can be stored on any of the volume's disks. If the a size of a merged part estimated to be bigger than `max_data_part_size_bytes` then this part will be written to a next volume. Basically this feature allows to keep new/small parts on a hot (SSD) volume and move them to a cold (HDD) volume when they reach large size. Do not use this setting if your policy has only one volume.
- `move_factor` — when the amount of available space gets lower than this factor, data automatically start to move on the next volume if any (by default, 0.1).
- `prefer_not_to_merge` — Disables merging of data parts on this volume. When this setting is enabled, merging data on this volume is not allowed. This allows controlling how ClickHouse works with slow disks.

Configuration examples:

```

<storage_configuration>
...
<policies>
  <hdd_in_order> <!-- policy name -->
    <volumes>
      <single> <!-- volume name -->
        <disk>disk1</disk>
        <disk>disk2</disk>
      </single>
    </volumes>
  </hdd_in_order>

  <moving_from_ss_to_hdd>
    <volumes>
      <hot>
        <disk>fast_ss</disk>
        <max_data_part_size_bytes>1073741824</max_data_part_size_bytes>
      </hot>
      <cold>
        <disk>disk1</disk>
      </cold>
    </volumes>
    <move_factor>0.2</move_factor>
  </moving_from_ss_to_hdd>

  <small_jbod_with_external_no_merges>
    <volumes>
      <main>
        <disk>jbod1</disk>
      </main>
      <external>
        <disk>external</disk>
        <prefer_not_to_merge>true</prefer_not_to_merge>
      </external>
    </volumes>
  </small_jbod_with_external_no_merges>
</policies>
...
</storage_configuration>

```

In given example, the `hdd_in_order` policy implements the **round-robin** approach. Thus this policy defines only one volume (single), the data parts are stored on all its disks in circular order. Such policy can be quite useful if there are several similar disks are mounted to the system, but RAID is not configured. Keep in mind that each individual disk drive is not reliable and you might want to compensate it with replication factor of 3 or more.

If there are different kinds of disks available in the system, `moving_from_ss_to_hdd` policy can be used instead. The volume `hot` consists of an SSD disk (`fast_ss`), and the maximum size of a part that can be stored on this volume is 1GB. All the parts with the size larger than 1GB will be stored directly on the `cold` volume, which contains an HDD disk `disk1`.

Also, once the disk `fast_ss` gets filled by more than 80%, data will be transferred to the `disk1` by a background process.

The order of volume enumeration within a storage policy is important. Once a volume is overfilled, data are moved to the next one. The order of disk enumeration is important as well because data are stored on them in turns.

When creating a table, one can apply one of the configured storage policies to it:

```
CREATE TABLE table_with_non_default_policy (
    EventDate Date,
    OrderID UInt64,
    BannerID UInt64,
    SearchPhrase String
) ENGINE = MergeTree
ORDER BY (OrderID, BannerID)
PARTITION BY toYYYYMM(EventDate)
SETTINGS storage_policy = 'moving_from_ss_to_hdd'
```

The default storage policy implies using only one volume, which consists of only one disk given in <path>. You could change storage policy after table creation with [ALTER TABLE ... MODIFY SETTING] query, new policy should include all old disks and volumes with same names.

The number of threads performing background moves of data parts can be changed by [background\\_move\\_pool\\_size](#) setting.

## Details

In the case of MergeTree tables, data is getting to disk in different ways:

- As a result of an insert ([INSERT](#) query).
- During background merges and [mutations](#).
- When downloading from another replica.
- As a result of partition freezing [ALTER TABLE ... FREEZE PARTITION](#).

In all these cases except for mutations and partition freezing, a part is stored on a volume and a disk according to the given storage policy:

1. The first volume (in the order of definition) that has enough disk space for storing a part (`unreserved_space > current_part_size`) and allows for storing parts of a given size (`max_data_part_size_bytes > current_part_size`) is chosen.
2. Within this volume, that disk is chosen that follows the one, which was used for storing the previous chunk of data, and that has free space more than the part size (`unreserved_space - keep_free_space_bytes > current_part_size`).

Under the hood, mutations and partition freezing make use of [hard links](#). Hard links between different disks are not supported, therefore in such cases the resulting parts are stored on the same disks as the initial ones.

In the background, parts are moved between volumes on the basis of the amount of free space (`move_factor` parameter) according to the order the volumes are declared in the configuration file.

Data is never transferred from the last one and into the first one. One may use system tables [system.part\\_log](#) (field type = `MOVE_PART`) and [system.parts](#) (fields `path` and `disk`) to monitor background moves. Also, the detailed information can be found in server logs.

User can force moving a part or a partition from one volume to another using the query [ALTER TABLE ... MOVE PART|PARTITION ... TO VOLUME|DISK ...](#), all the restrictions for background operations are taken into account. The query initiates a move on its own and does not wait for background operations to be completed. User will get an error message if not enough free space is available or if any of the required conditions are not met.

Moving data does not interfere with data replication. Therefore, different storage policies can be specified for the same table on different replicas.

After the completion of background merges and mutations, old parts are removed only after a certain amount of time (`old_parts_lifetime`).

During this time, they are not moved to other volumes or disks. Therefore, until the parts are finally removed, they are still taken into account for evaluation of the occupied disk space.

## Using S3 for Data Storage

MergeTree family table engines can store data to **S3** using a disk with type `s3`.

This feature is under development and not ready for production. There are known drawbacks such as very low performance.

Configuration markup:

```
<storage_configuration>
...
<disks>
  <s3>
    <type>s3</type>
    <endpoint>https://storage.yandexcloud.net/my-bucket/root-path/</endpoint>
    <access_key_id>your_access_key_id</access_key_id>
    <secret_access_key>your_secret_access_key</secret_access_key>
    <region></region>

  <server_side_encryption_customer_key_base64>your_base64_encoded_customer_key</server_side_encryption_customer_key_base64>
    <proxy>
      <uri>http://proxy1</uri>
      <uri>http://proxy2</uri>
    </proxy>
    <connect_timeout_ms>10000</connect_timeout_ms>
    <request_timeout_ms>5000</request_timeout_ms>
    <retry_attempts>10</retry_attempts>
    <single_read_retries>4</single_read_retries>
    <min_bytes_for_seek>1000</min_bytes_for_seek>
    <metadata_path>/var/lib/clickhouse/disks/s3/</metadata_path>
    <cache_enabled>true</cache_enabled>
    <cache_path>/var/lib/clickhouse/disks/s3/cache/</cache_path>
    <skip_access_check>false</skip_access_check>
  </s3>
</disks>
...
</storage_configuration>
```

Required parameters:

- `endpoint` — S3 endpoint URL in `path` or `virtual hosted styles`. Endpoint URL should contain a bucket and root path to store data.
- `access_key_id` — S3 access key id.
- `secret_access_key` — S3 secret access key.

Optional parameters:

- `region` — S3 region name.
- `use_environment_credentials` — Reads AWS credentials from the Environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` and `AWS_SESSION_TOKEN` if they exist. Default value is `false`.
- `use_insecure_imds_request` — If set to `true`, S3 client will use insecure IMDS request while obtaining credentials from Amazon EC2 metadata. Default value is `false`.

- `proxy` — Proxy configuration for S3 endpoint. Each `uri` element inside `proxy` block should contain a proxy URL.
- `connect_timeout_ms` — Socket connect timeout in milliseconds. Default value is `10 seconds`.
- `request_timeout_ms` — Request timeout in milliseconds. Default value is `5 seconds`.
- `retry_attempts` — Number of retry attempts in case of failed request. Default value is `10`.
- `single_read_retries` — Number of retry attempts in case of connection drop during read. Default value is `4`.
- `min_bytes_for_seek` — Minimal number of bytes to use seek operation instead of sequential read. Default value is `1 Mb`.
- `metadata_path` — Path on local FS to store metadata files for S3. Default value is `/var/lib/clickhouse/disks/<disk_name>/`.
- `cache_enabled` — Allows to cache mark and index files on local FS. Default value is `true`.
- `cache_path` — Path on local FS where to store cached mark and index files. Default value is `/var/lib/clickhouse/disks/<disk_name>/cache/`.
- `skip_access_check` — If true, disk access checks will not be performed on disk start-up. Default value is `false`.
- `server_side_encryption_customer_key_base64` — If specified, required headers for accessing S3 objects with SSE-C encryption will be set.

S3 disk can be configured as `main` or `cold` storage:

```
<storage_configuration>
...
<disks>
  <s3>
    <type>s3</type>
    <endpoint>https://storage.yandexcloud.net/my-bucket/root-path/</endpoint>
    <access_key_id>your_access_key_id</access_key_id>
    <secret_access_key>your_secret_access_key</secret_access_key>
  </s3>
</disks>
<policies>
  <s3_main>
    <volumes>
      <main>
        <disk>s3</disk>
      </main>
    </volumes>
  </s3_main>
  <s3_cold>
    <volumes>
      <main>
        <disk>default</disk>
      </main>
      <external>
        <disk>s3</disk>
      </external>
    </volumes>
    <move_factor>0.2</move_factor>
  </s3_cold>
</policies>
...
</storage_configuration>
```

In case of `cold` option a data can be moved to S3 if local disk free size will be smaller than `move_factor * disk_size` or by TTL move rule.

# Data Replication

Replication is only supported for tables in the MergeTree family:

- ReplicatedMergeTree
- ReplicatedSummingMergeTree
- ReplicatedReplacingMergeTree
- ReplicatedAggregatingMergeTree
- ReplicatedCollapsingMergeTree
- ReplicatedVersionedCollapsingMergeTree
- ReplicatedGraphiteMergeTree

Replication works at the level of an individual table, not the entire server. A server can store both replicated and non-replicated tables at the same time.

Replication does not depend on sharding. Each shard has its own independent replication.

Compressed data for `INSERT` and `ALTER` queries is replicated (for more information, see the documentation for `ALTER`).

`CREATE`, `DROP`, `ATTACH`, `DETACH` and `RENAME` queries are executed on a single server and are not replicated:

- The `CREATE TABLE` query creates a new replicatable table on the server where the query is run. If this table already exists on other servers, it adds a new replica.
- The `DROP TABLE` query deletes the replica located on the server where the query is run.
- The `RENAME` query renames the table on one of the replicas. In other words, replicated tables can have different names on different replicas.

ClickHouse uses [Apache ZooKeeper](#) for storing replicas meta information. Use ZooKeeper version 3.4.5 or newer.

To use replication, set parameters in the `zookeeper` server configuration section.

## Attention

Don't neglect the security setting. ClickHouse supports the digest [\*\*ACL scheme\*\*](#) of the ZooKeeper security subsystem.

Example of setting the addresses of the ZooKeeper cluster:

```
<zookeeper>
  <node>
    <host>example1</host>
    <port>2181</port>
  </node>
  <node>
    <host>example2</host>
    <port>2181</port>
  </node>
  <node>
    <host>example3</host>
    <port>2181</port>
  </node>
</zookeeper>
```

ClickHouse also supports to store replicas meta information in the auxiliary ZooKeeper cluster by providing ZooKeeper cluster name and path as engine arguments.

In other word, it supports to store the metadata of differnt tables in different ZooKeeper clusters.

Example of setting the addresses of the auxiliary ZooKeeper cluster:

```
<auxiliary_zookeepers>
  <zookeeper2>
    <node>
      <host>example_2_1</host>
      <port>2181</port>
    </node>
    <node>
      <host>example_2_2</host>
      <port>2181</port>
    </node>
    <node>
      <host>example_2_3</host>
      <port>2181</port>
    </node>
  </zookeeper2>
  <zookeeper3>
    <node>
      <host>example_3_1</host>
      <port>2181</port>
    </node>
  </zookeeper3>
</auxiliary_zookeepers>
```

To store table datameta in a auxiliary ZooKeeper cluster instead of default ZooKeeper cluster, we can use the SQL to create table with

ReplicatedMergeTree engine as follow:

```
CREATE TABLE table_name ( ... ) ENGINE =
ReplicatedMergeTree('zookeeper_name_configured_in_auxiliary_zookeepers:path', 'replica_name') ...
```

You can specify any existing ZooKeeper cluster and the system will use a directory on it for its own data (the directory is specified when creating a replicatable table).

If ZooKeeper isn't set in the config file, you can't create replicated tables, and any existing replicated tables will be read-only.

ZooKeeper is not used in `SELECT` queries because replication does not affect the performance of `SELECT` and queries run just as fast as they do for non-replicated tables. When querying distributed replicated tables, ClickHouse behavior is controlled by the settings `max_replica_delay_for_distributed_queries` and `fallback_to_stale_replicas_for_distributed_queries`.

For each `INSERT` query, approximately ten entries are added to ZooKeeper through several transactions. (To be more precise, this is for each inserted block of data; an `INSERT` query contains one block or one block per `max_insert_block_size = 1048576` rows.) This leads to slightly longer latencies for `INSERT` compared to non-replicated tables. But if you follow the recommendations to insert data in batches of no more than one `INSERT` per second, it does not create any problems. The entire ClickHouse cluster used for coordinating one ZooKeeper cluster has a total of several hundred `INSERTs` per second. The throughput on data inserts (the number of rows per second) is just as high as for non-replicated data.

For very large clusters, you can use different ZooKeeper clusters for different shards. However, this hasn't proven necessary on the Yandex.Metrica cluster (approximately 300 servers).

Replication is asynchronous and multi-master. `INSERT` queries (as well as `ALTER`) can be sent to any available server. Data is inserted on the server where the query is run, and then it is copied to the other servers. Because it is asynchronous, recently inserted data appears on the other replicas with some latency. If part of the replicas are not available, the data is written when they become available. If a replica is available, the latency is the amount of time it takes to transfer the block of compressed data over the network. The number of threads performing background tasks for replicated tables can be set by `background_schedule_pool_size` setting.

`ReplicatedMergeTree` engine uses a separate thread pool for replicated fetches. Size of the pool is limited by the `background_fetches_pool_size` setting which can be tuned with a server restart.

By default, an `INSERT` query waits for confirmation of writing the data from only one replica. If the data was successfully written to only one replica and the server with this replica ceases to exist, the stored data will be lost. To enable getting confirmation of data writes from multiple replicas, use the `insert_quorum` option.

Each block of data is written atomically. The `INSERT` query is divided into blocks up to `max_insert_block_size = 1048576` rows. In other words, if the `INSERT` query has less than 1048576 rows, it is made atomically.

Data blocks are deduplicated. For multiple writes of the same data block (data blocks of the same size containing the same rows in the same order), the block is only written once. The reason for this is in case of network failures when the client application does not know if the data was written to the DB, so the `INSERT` query can simply be repeated. It does not matter which replica `INSERTs` were sent to with identical data. `INSERTs` are idempotent. Deduplication parameters are controlled by `merge_tree` server settings.

During replication, only the source data to insert is transferred over the network. Further data transformation (merging) is coordinated and performed on all the replicas in the same way. This minimizes network usage, which means that replication works well when replicas reside in different datacenters. (Note that duplicating data in different datacenters is the main goal of replication.)

You can have any number of replicas of the same data. Yandex.Metrica uses double replication in production. Each server uses RAID-5 or RAID-6, and RAID-10 in some cases. This is a relatively reliable and convenient solution.

The system monitors data synchronicity on replicas and is able to recover after a failure. Failover is automatic (for small differences in data) or semi-automatic (when data differs too much, which may indicate a configuration error).

## Creating Replicated Tables

The `Replicated` prefix is added to the table engine name. For example: `ReplicatedMergeTree`.

### **Replicated\*MergeTree parameters**

- `zoo_path` — The path to the table in ZooKeeper.
- `replica_name` — The replica name in ZooKeeper.

- `other_parameters` — Parameters of an engine which is used for creating the replicated version, for example, `version` in `ReplacingMergeTree`.

Example:

```
CREATE TABLE table_name
(
    EventDate DateTime,
    CounterID UInt32,
    UserID UInt32,
    ver UInt16
) ENGINE = ReplicatedReplacingMergeTree('/clickhouse/tables/{layer}-{shard}/table_name', '{replica}', ver)
PARTITION BY toYYYYMM(EventDate)
ORDER BY (CounterID, EventDate, intHash32(UserID))
SAMPLE BY intHash32(UserID);
```

#### ► Example in deprecated syntax

As the example shows, these parameters can contain substitutions in curly brackets. The substituted values are taken from the `macros` section of the configuration file.

Example:

```
<macros>
    <layer>05</layer>
    <shard>02</shard>
    <replica>example05-02-1.yandex.ru</replica>
</macros>
```

The path to the table in ZooKeeper should be unique for each replicated table. Tables on different shards should have different paths.

In this case, the path consists of the following parts:

`/clickhouse/tables/` is the common prefix. We recommend using exactly this one.

`{layer}-{shard}` is the shard identifier. In this example it consists of two parts, since the Yandex.Metrica cluster uses bi-level sharding. For most tasks, you can leave just the `{shard}` substitution, which will be expanded to the shard identifier.

`table_name` is the name of the node for the table in ZooKeeper. It is a good idea to make it the same as the table name. It is defined explicitly, because in contrast to the table name, it does not change after a `RENAME` query.

*HINT:* you could add a database name in front of `table_name` as well. E.g. `db_name.table_name`

The two built-in substitutions `{database}` and `{table}` can be used, they expand into the table name and the database name respectively (unless these macros are defined in the `macros` section). So the zookeeper path can be specified as `'/clickhouse/tables/{layer}-{shard}/{database}/{table}'`.

Be careful with table renames when using these built-in substitutions. The path in Zookeeper cannot be changed, and when the table is renamed, the macros will expand into a different path, the table will refer to a path that does not exist in Zookeeper, and will go into read-only mode.

The replica name identifies different replicas of the same table. You can use the server name for this, as in the example. The name only needs to be unique within each shard.

You can define the parameters explicitly instead of using substitutions. This might be convenient for testing and for configuring small clusters. However, you can't use distributed DDL queries (ON CLUSTER) in this case.

When working with large clusters, we recommend using substitutions because they reduce the probability of error.

You can specify default arguments for `Replicated` table engine in the server configuration file. For instance:

```
<default_replica_path>/clickhouse/tables/{shard}/{database}/{table}</default_replica_path>
<default_replica_name>{replica}</default_replica_name>
```

In this case, you can omit arguments when creating tables:

```
CREATE TABLE table_name (
    x UInt32
) ENGINE = ReplicatedMergeTree
ORDER BY x;
```

It is equivalent to:

```
CREATE TABLE table_name (
    x UInt32
) ENGINE = ReplicatedMergeTree('/clickhouse/tables/{shard}/{database}/table_name', '{replica}')
ORDER BY x;
```

Run the `CREATE TABLE` query on each replica. This query creates a new replicated table, or adds a new replica to an existing one.

If you add a new replica after the table already contains some data on other replicas, the data will be copied from the other replicas to the new one after running the query. In other words, the new replica syncs itself with the others.

To delete a replica, run `DROP TABLE`. However, only one replica is deleted – the one that resides on the server where you run the query.

## Recovery After Failures

If ZooKeeper is unavailable when a server starts, replicated tables switch to read-only mode. The system periodically attempts to connect to ZooKeeper.

If ZooKeeper is unavailable during an `INSERT`, or an error occurs when interacting with ZooKeeper, an exception is thrown.

After connecting to ZooKeeper, the system checks whether the set of data in the local file system matches the expected set of data (ZooKeeper stores this information). If there are minor inconsistencies, the system resolves them by syncing data with the replicas.

If the system detects broken data parts (with the wrong size of files) or unrecognized parts (parts written to the file system but not recorded in ZooKeeper), it moves them to the `detached` subdirectory (they are not deleted). Any missing parts are copied from the replicas.

Note that ClickHouse does not perform any destructive actions such as automatically deleting a large amount of data.

When the server starts (or establishes a new session with ZooKeeper), it only checks the quantity and sizes of all files. If the file sizes match but bytes have been changed somewhere in the middle, this is not detected immediately, but only when attempting to read the data for a `SELECT` query. The query throws an exception about a non-matching checksum or size of a compressed block. In this case, data parts are added to the verification queue and copied from the replicas if necessary.

If the local set of data differs too much from the expected one, a safety mechanism is triggered. The server enters this in the log and refuses to launch. The reason for this is that this case may indicate a configuration error, such as if a replica on a shard was accidentally configured like a replica on a different shard. However, the thresholds for this mechanism are set fairly low, and this situation might occur during normal failure recovery. In this case, data is restored semi-automatically - by “pushing a button”.

To start recovery, create the node `/path_to_table/replica_name flags/force_restore_data` in ZooKeeper with any content, or run the command to restore all replicated tables:

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

Then restart the server. On start, the server deletes these flags and starts recovery.

## Recovery After Complete Data Loss

If all data and metadata disappeared from one of the servers, follow these steps for recovery:

1. Install ClickHouse on the server. Define substitutions correctly in the config file that contains the shard identifier and replicas, if you use them.
2. If you had unreplicated tables that must be manually duplicated on the servers, copy their data from a replica (in the directory `/var/lib/clickhouse/data/db_name/table_name/`).
3. Copy table definitions located in `/var/lib/clickhouse/metadata/` from a replica. If a shard or replica identifier is defined explicitly in the table definitions, correct it so that it corresponds to this replica. (Alternatively, start the server and make all the `ATTACH TABLE` queries that should have been in the `.sql` files in `/var/lib/clickhouse/metadata/`.)
4. To start recovery, create the ZooKeeper node `/path_to_table/replica_name flags/force_restore_data` with any content, or run the command to restore all replicated tables: `sudo -u clickhouse touch /var/lib/clickhouse/flags/force_restore_data`

Then start the server (restart, if it is already running). Data will be downloaded from replicas.

An alternative recovery option is to delete information about the lost replica from ZooKeeper (`/path_to_table/replica_name`), then create the replica again as described in “[Creating replicated tables](#)”.

There is no restriction on network bandwidth during recovery. Keep this in mind if you are restoring many replicas at once.

## Converting from MergeTree to ReplicatedMergeTree

We use the term `MergeTree` to refer to all table engines in the `MergeTree` family, the same as for `ReplicatedMergeTree`.

If you had a `MergeTree` table that was manually replicated, you can convert it to a replicated table. You might need to do this if you have already collected a large amount of data in a `MergeTree` table and now you want to enable replication.

If the data differs on various replicas, first sync it, or delete this data on all the replicas except one.

Rename the existing `MergeTree` table, then create a `ReplicatedMergeTree` table with the old name. Move the data from the old table to the `detached` subdirectory inside the directory with the new table data (`/var/lib/clickhouse/data/db_name/table_name/`).

Then run `ALTER TABLE ATTACH PARTITION` on one of the replicas to add these data parts to the working set.

## Converting from ReplicatedMergeTree to MergeTree

Create a MergeTree table with a different name. Move all the data from the directory with the ReplicatedMergeTree table data to the new table's data directory. Then delete the ReplicatedMergeTree table and restart the server.

If you want to get rid of a ReplicatedMergeTree table without launching the server:

- Delete the corresponding .sql file in the metadata directory (`/var/lib/clickhouse/metadata/`).
- Delete the corresponding path in ZooKeeper (`/path_to_table/replica_name`).

After this, you can launch the server, create a MergeTree table, move the data to its directory, and then restart the server.

## Recovery When Metadata in the Zookeeper Cluster Is Lost or Damaged

If the data in ZooKeeper was lost or damaged, you can save data by moving it to an unreplicated table as described above.

### See Also

- [background\\_schedule\\_pool\\_size](#)
- [background\\_fetches\\_pool\\_size](#)
- [execute\\_merges\\_on\\_single\\_replica\\_time\\_threshold](#)
- [max\\_replicated\\_fetches\\_network\\_bandwidth](#)
- [max\\_replicated\\_sends\\_network\\_bandwidth](#)

## Custom Partitioning Key

### Warning

In most cases you don't need partition key, and in most other cases you don't need partition key more granular than by months. Partitioning does not speed up queries (in contrast to the ORDER BY expression). You should never use too granular partitioning. Don't partition your data by client identifiers or names (instead make client identifier or name the first column in the ORDER BY expression).

Partitioning is available for the MergeTree family tables (including replicated tables). Materialized views based on MergeTree tables support partitioning, as well.

A partition is a logical combination of records in a table by a specified criterion. You can set a partition by an arbitrary criterion, such as by month, by day, or by event type. Each partition is stored separately to simplify manipulations of this data. When accessing the data, ClickHouse uses the smallest subset of partitions possible.

The partition is specified in the PARTITION BY expr clause when [creating a table](#). The partition key can be any expression from the table columns. For example, to specify partitioning by month, use the expression `toYYYYMM(date_column)`:

```

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

```

The partition key can also be a tuple of expressions (similar to the [primary key](#)). For example:

```

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

```

In this example, we set partitioning by the event types that occurred during the current week.

By default, the floating-point partition key is not supported. To use it enable the setting [allow\\_floating\\_point\\_partition\\_key](#).

When inserting new data to a table, this data is stored as a separate part (chunk) sorted by the primary key. In 10-15 minutes after inserting, the parts of the same partition are merged into the entire part.

## Info

A merge only works for data parts that have the same value for the partitioning expression. This means **you shouldn't make overly granular partitions** (more than about a thousand partitions). Otherwise, the `SELECT` query performs poorly because of an unreasonably large number of files in the file system and open file descriptors.

Use the `system.parts` table to view the table parts and partitions. For example, let's assume that we have a `visits` table with partitioning by month. Let's perform the `SELECT` query for the `system.parts` table:

```

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

The `partition` column contains the names of the partitions. There are two partitions in this example: `201901` and `201902`. You can use this column value to specify the partition name in [ALTER ... PARTITION](#) queries.

The `name` column contains the names of the partition data parts. You can use this column to specify the name of the part in the [ALTER ATTACH PART](#) query.

Let's break down the name of the first part: `201901_1_3_1`:

- 201901 is the partition name.
- 1 is the minimum number of the data block.
- 3 is the maximum number of the data block.
- 1 is the chunk level (the depth of the merge tree it is formed from).

## Info

The parts of old-type tables have the name: 20190117\_20190123\_2\_2\_0 (minimum date - maximum date - minimum block number - maximum block number - level).

The active column shows the status of the part. 1 is active; 0 is inactive. The inactive parts are, for example, source parts remaining after merging to a larger part. The corrupted data parts are also indicated as inactive.

As you can see in the example, there are several separated parts of the same partition (for example, 201901\_1\_3\_1 and 201901\_1\_9\_2). This means that these parts are not merged yet. ClickHouse merges the inserted parts of data periodically, approximately 15 minutes after inserting. In addition, you can perform a non-scheduled merge using the **OPTIMIZE** query. Example:

```
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

Inactive parts will be deleted approximately 10 minutes after merging.

Another way to view a set of parts and partitions is to go into the directory of the table: /var/lib/clickhouse/data/<database>/<table>/. For example:

```
/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
```

The folders '201901\_1\_1\_0', '201901\_1\_7\_1' and so on are the directories of the parts. Each part relates to a corresponding partition and contains data just for a certain month (the table in this example has partitioning by month).

The `detached` directory contains parts that were detached from the table using the `DETACH` query. The corrupted parts are also moved to this directory, instead of being deleted. The server does not use the parts from the `detached` directory. You can add, delete, or modify the data in this directory at any time – the server will not know about this until you run the `ATTACH` query.

Note that on the operating server, you cannot manually change the set of parts or their data on the file system, since the server will not know about it. For non-replicated tables, you can do this when the server is stopped, but it isn't recommended. For replicated tables, the set of parts cannot be changed in any case.

ClickHouse allows you to perform operations with the partitions: delete them, copy from one table to another, or create a backup. See the list of all operations in the section [Manipulations With Partitions and Parts](#).

## ReplacingMergeTree

The engine differs from `MergeTree` in that it removes duplicate entries with the same `sorting key` value (`ORDER BY` table section, not `PRIMARY KEY`).

Data deduplication occurs only during a merge. Merging occurs in the background at an unknown time, so you can't plan for it. Some of the data may remain unprocessed. Although you can run an unscheduled merge using the `OPTIMIZE` query, do not count on using it, because the `OPTIMIZE` query will read and write a large amount of data.

Thus, `ReplacingMergeTree` is suitable for clearing out duplicate data in the background in order to save space, but it does not guarantee the absence of duplicates.

## Creating a Table

```
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]
[PRIMARY KEY expr]
[SAMPLE BY expr]
[SETTINGS name=value, ...]
```

For a description of request parameters, see [statement description](#).

### Attention

Uniqueness of rows is determined by the `ORDER BY` table section, not `PRIMARY KEY`.

### ReplacingMergeTree Parameters

- `ver` — column with the version number. Type `UInt*`, `Date`, `DateTime` or `DateTime64`. Optional parameter.

When merging, `ReplacingMergeTree` from all the rows with the same sorting key leaves only one:

- The last in the selection, if `ver` not set. A selection is a set of rows in a set of parts participating in the merge. The most recently created part (the last insert) will be the last one in the selection. Thus, after deduplication, the very last row from the most recent insert will remain for each unique sorting key.
- With the maximum version, if `ver` specified.

## Query clauses

When creating a `ReplacingMergeTree` table the same **clauses** are required, as when creating a `MergeTree` table.

### ► Deprecated Method for Creating a Table

# SummingMergeTree

The engine inherits from `MergeTree`. The difference is that when merging data parts for `SummingMergeTree` tables ClickHouse replaces all the rows with the same primary key (or more accurately, with the same **sorting key**) with one row which contains summarized values for the columns with the numeric data type. If the sorting key is composed in a way that a single key value corresponds to large number of rows, this significantly reduces storage volume and speeds up data selection.

We recommend using the engine together with `MergeTree`. Store complete data in `MergeTree` table, and use `SummingMergeTree` for aggregated data storing, for example, when preparing reports. Such an approach will prevent you from losing valuable data due to an incorrectly composed primary key.

## Creating a Table

```
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, ...]
```

For a description of request parameters, see [request description](#).

### Parameters of SummingMergeTree

- `columns` - a tuple with the names of columns where values will be summarized. Optional parameter. The columns must be of a numeric type and must not be in the primary key.

If `columns` not specified, ClickHouse summarizes the values in all columns with a numeric data type that are not in the primary key.

## Query clauses

When creating a `SummingMergeTree` table the same **clauses** are required, as when creating a `MergeTree` table.

### ► Deprecated Method for Creating a Table

# Usage Example

Consider the following table:

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

Insert data to it:

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

ClickHouse may sum all the rows not completely ([see below](#)), so we use an aggregate function `sum` and `GROUP BY` clause in the query.

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

key	sum(value)
2	1
1	3

## Data Processing

When data are inserted into a table, they are saved as-is. ClickHouse merges the inserted parts of data periodically and this is when rows with the same primary key are summed and replaced with one for each resulting part of data.

ClickHouse can merge the data parts so that different resulting parts of data can consist rows with the same primary key, i.e. the summation will be incomplete. Therefore (`SELECT`) an aggregate function `sum()` and `GROUP BY` clause should be used in a query as described in the example above.

## Common Rules for Summation

The values in the columns with the numeric data type are summarized. The set of columns is defined by the parameter `columns`.

If the values were 0 in all of the columns for summation, the row is deleted.

If column is not in the primary key and is not summarized, an arbitrary value is selected from the existing ones.

The values are not summarized for columns in the primary key.

## The Summation in the Aggregatefunction Columns

For columns of [AggregateFunction type](#) ClickHouse behaves as [AggregatingMergeTree](#) engine aggregating according to the function.

## Nested Structures

Table can have nested data structures that are processed in a special way.

If the name of a nested table ends with `Map` and it contains at least two columns that meet the following criteria:

- the first column is numeric (`*Int*`, `Date`, `DateTime`) or a string (`String`, `FixedString`), let's call it `key`,

- the other columns are arithmetic (\*Int\*, Float32/64), let's call it (values...),

then this nested table is interpreted as a mapping of key => (values...), and when merging its rows, the elements of two data sets are merged by key with a summation of the corresponding (values...).

Examples:

```
[(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)]
```

When requesting data, use the `sumMap(key, value)` function for aggregation of Map.

For nested data structure, you do not need to specify its columns in the tuple of columns for summation.

## AggregatingMergeTree

The engine inherits from [MergeTree](#), altering the logic for data parts merging. ClickHouse replaces all rows with the same primary key (or more accurately, with the same [sorting key](#)) with a single row (within a one data part) that stores a combination of states of aggregate functions.

You can use `AggregatingMergeTree` tables for incremental data aggregation, including for aggregated materialized views.

The engine processes all columns with the following types:

- [AggregateFunction](#)
- [SimpleAggregateFunction](#)

It is appropriate to use `AggregatingMergeTree` if it reduces the number of rows by orders.

## Creating a Table

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
(
    name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1],
    name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2],
    ...
) ENGINE = AggregatingMergeTree()
[PARTITION BY expr]
[ORDER BY expr]
[SAMPLE BY expr]
[TTL expr]
[SETTINGS name=value, ...]
```

For a description of request parameters, see [request description](#).

### Query clauses

When creating a `AggregatingMergeTree` table the same [clauses](#) are required, as when creating a `MergeTree` table.

#### ► **Deprecated Method for Creating a Table**

## SELECT and INSERT

To insert data, use `INSERT SELECT` query with aggregate -State- functions.

When selecting data from `AggregatingMergeTree` table, use `GROUP BY` clause and the same aggregate functions as when inserting data, but using `-Merge` suffix.

In the results of `SELECT` query, the values of `AggregateFunction` type have implementation-specific binary representation for all of the ClickHouse output formats. If dump data into, for example, `TabSeparated` format with `SELECT` query then this dump can be loaded back using `INSERT` query.

## Example of an Aggregated Materialized View

`AggregatingMergeTree` materialized view that watches the `test.visits` table:

```
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;
```

Inserting data into the `test.visits` table.

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

The data are inserted in both the table and view `test.basic` that will perform the aggregation.

To get the aggregated data, we need to execute a query such as `SELECT ... GROUP BY ...` from the view `test.basic`:

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

## CollapsingMergeTree

The engine inherits from `MergeTree` and adds the logic of rows collapsing to data parts merge algorithm.

`CollapsingMergeTree` asynchronously deletes (collapses) pairs of rows if all of the fields in a sorting key (`ORDER BY`) are equivalent excepting the particular field `Sign` which can have `1` and `-1` values. Rows without a pair are kept. For more details see the [Collapsing](#) section of the document.

The engine may significantly reduce the volume of storage and increase the efficiency of `SELECT` query as a consequence.

## Creating a Table

```

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, ...]

```

For a description of query parameters, see [query description](#).

### **CollapsingMergeTree Parameters**

- **sign** — Name of the column with the type of row: 1 is a “state” row, -1 is a “cancel” row.

Column data type — Int8.

### **Query clauses**

When creating a CollapsingMergeTree table, the same [query clauses](#) are required, as when creating a MergeTree table.

#### ► **Deprecated Method for Creating a Table**

## **Collapsing**

### **Data**

Consider the situation where you need to save continually changing data for some object. It sounds logical to have one row for an object and update it at any change, but update operation is expensive and slow for DBMS because it requires rewriting of the data in the storage. If you need to write data quickly, update not acceptable, but you can write the changes of an object sequentially as follows.

Use the particular column **Sign**. If **Sign = 1** it means that the row is a state of an object, let's call it “state” row. If **Sign = -1** it means the cancellation of the state of an object with the same attributes, let's call it “cancel” row.

For example, we want to calculate how much pages users checked at some site and how long they were there. At some moment we write the following row with the state of user activity:

UserID	PageViews	Duration	Sign
4324182021466249494	5	146	1

At some moment later we register the change of user activity and write it with the following two rows.

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

The first row cancels the previous state of the object (user). It should copy the sorting key fields of the cancelled state excepting **Sign**.

The second row contains the current state.

As we need only the last state of user activity, the rows

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

can be deleted collapsing the invalid (old) state of an object. CollapsingMergeTree does this while merging of the data parts.

Why we need 2 rows for each change read in the [Algorithm](#) paragraph.

### Peculiar properties of such approach

1. The program that writes the data should remember the state of an object to be able to cancel it. “Cancel” string should contain copies of the sorting key fields of the “state” string and the opposite Sign. It increases the initial size of storage but allows to write the data quickly.
2. Long growing arrays in columns reduce the efficiency of the engine due to load for writing. The more straightforward data, the higher the efficiency.
3. The SELECT results depend strongly on the consistency of object changes history. Be accurate when preparing data for inserting. You can get unpredictable results in inconsistent data, for example, negative values for non-negative metrics such as session depth.

## Algorithm

When ClickHouse merges data parts, each group of consecutive rows with the same sorting key (ORDER BY) is reduced to not more than two rows, one with Sign = 1 (“state” row) and another with Sign = -1 (“cancel” row). In other words, entries collapse.

For each resulting data part ClickHouse saves:

1. The first “cancel” and the last “state” rows, if the number of “state” and “cancel” rows matches and the last row is a “state” row.
2. The last “state” row, if there are more “state” rows than “cancel” rows.
3. The first “cancel” row, if there are more “cancel” rows than “state” rows.
4. None of the rows, in all other cases.

Also when there are at least 2 more “state” rows than “cancel” rows, or at least 2 more “cancel” rows than “state” rows, the merge continues, but ClickHouse treats this situation as a logical error and records it in the server log. This error can occur if the same data were inserted more than once.

Thus, collapsing should not change the results of calculating statistics.

Changes gradually collapsed so that in the end only the last state of almost every object left.

The Sign is required because the merging algorithm does not guarantee that all of the rows with the same sorting key will be in the same resulting data part and even on the same physical server. ClickHouse process SELECT queries with multiple threads, and it can not predict the order of rows in the result. The aggregation is required if there is a need to get completely “collapsed” data from CollapsingMergeTree table.

To finalize collapsing, write a query with GROUP BY clause and aggregate functions that account for the sign. For example, to calculate quantity, use sum(Sign) instead of count(). To calculate the sum of something, use sum(Sign \* x) instead of sum(x), and so on, and also add HAVING sum(Sign) > 0.

The aggregates count, sum and avg could be calculated this way. The aggregate uniq could be calculated if an object has at least one state not collapsed. The aggregates min and max could not be calculated because CollapsingMergeTree does not save the values history of the collapsed states.

If you need to extract data without aggregation (for example, to check whether rows are present whose newest values match certain conditions), you can use the FINAL modifier for the FROM clause. This approach is significantly less efficient.

# Example of Use

Example data:

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

Creation of the table:

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

Insertion of the data:

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

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

We use two `INSERT` queries to create two different data parts. If we insert the data with one query ClickHouse creates one data part and will not perform any merge ever.

Getting the data:

```
SELECT * FROM UAct
```

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

UserID	PageViews	Duration	Sign
4324182021466249494	5	146	1

What do we see and where is collapsing?

With two `INSERT` queries, we created 2 data parts. The `SELECT` query was performed in 2 threads, and we got a random order of rows. Collapsing not occurred because there was no merge of the data parts yet. ClickHouse merges data part in an unknown moment which we can not predict.

Thus we need aggregation:

```
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

If we do not need aggregation and want to force collapsing, we can use `FINAL` modifier for `FROM` clause.

```
SELECT * FROM UAct FINAL
```

UserID	PageViews	Duration	Sign
4324182021466249494	6	185	1

This way of selecting the data is very inefficient. Don't use it for big tables.

## Example of Another Approach

Example data:

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

The idea is that merges take into account only key fields. And in the "Cancel" line we can specify negative values that equalize the previous version of the row when summing without using the `Sign` column. For this approach, it is necessary to change the data type `PageViews`, `Duration` to store negative values of `UInt8` -> `Int16`.

```
CREATE TABLE UAct
(
    UserID UInt64,
    PageViews Int16,
    Duration Int16,
    Sign Int8
)
ENGINE = CollapsingMergeTree(Sign)
ORDER BY UserID
```

Let's test the approach:

```
insert into UAct values(4324182021466249494, 5, 146, 1);
insert into UAct values(4324182021466249494, -5, -146, -1);
insert into UAct values(4324182021466249494, 6, 185, 1);

select * from UAct final; // avoid using final in production (just for a test or small tables)
```

UserID	PageViews	Duration	Sign
4324182021466249494	6	185	1

```
SELECT
    UserID,
    sum(PageViews) AS PageViews,
    sum(Duration) AS Duration
FROM UAct
GROUP BY UserID
```

UserID	PageViews	Duration
4324182021466249494	6	185

```
select count() FROM UAct
```

count()
3

```
optimize table UAct final;
```

```
select * FROM UAct
```

UserID	PageViews	Duration	Sign
4324182021466249494	6	185	1

## VersionedCollapsingMergeTree

This engine:

- Allows quick writing of object states that are continually changing.
- Deletes old object states in the background. This significantly reduces the volume of storage.

See the section [Collapsing](#) for details.

The engine inherits from [MergeTree](#) and adds the logic for collapsing rows to the algorithm for merging data parts. `VersionedCollapsingMergeTree` serves the same purpose as [CollapsingMergeTree](#) but uses a different collapsing algorithm that allows inserting the data in any order with multiple threads. In particular, the `Version` column helps to collapse the rows properly even if they are inserted in the wrong order. In contrast, [CollapsingMergeTree](#) allows only strictly consecutive insertion.

## Creating a Table

```
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, ...]
```

For a description of query parameters, see the [query description](#).

### Engine Parameters

VersionedCollapsingMergeTree(sign, version)
---

- **sign** — Name of the column with the type of row: **1** is a “state” row, **-1** is a “cancel” row.

The column data type should be `Int8`.

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

The column data type should be `UInt*`.

## Query Clauses

When creating a `VersionedCollapsingMergeTree` table, the same **clauses** are required as when creating a `MergeTree` table.

### ► Deprecated Method for Creating a Table

## Collapsing

### Data

Consider a situation where you need to save continually changing data for some object. It is reasonable to have one row for an object and update the row whenever there are changes. However, the update operation is expensive and slow for a DBMS because it requires rewriting the data in the storage. Update is not acceptable if you need to write data quickly, but you can write the changes to an object sequentially as follows.

Use the `Sign` column when writing the row. If `Sign = 1` it means that the row is a state of an object (let's call it the “state” row). If `Sign = -1` it indicates the cancellation of the state of an object with the same attributes (let's call it the “cancel” row). Also use the `Version` column, which should identify each state of an object with a separate number.

For example, we want to calculate how many pages users visited on some site and how long they were there. At some point in time we write the following row with the state of user activity:

UserID	PageViews	Duration	Sign	Version
4324182021466249494	5	146	1	1

At some point later we register the change of user activity and write it with the following two rows.

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

The first row cancels the previous state of the object (user). It should copy all of the fields of the canceled state except `Sign`.

The second row contains the current state.

Because we need only the last state of user activity, the rows

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

can be deleted, collapsing the invalid (old) state of the object. `VersionedCollapsingMergeTree` does this while merging the data parts.

To find out why we need two rows for each change, see [Algorithm](#).

## Notes on Usage

1. The program that writes the data should remember the state of an object to be able to cancel it. “Cancel” string should contain copies of the primary key fields and the version of the “state” string and the opposite `Sign`. It increases the initial size of storage but allows to write the data quickly.
2. Long growing arrays in columns reduce the efficiency of the engine due to the load for writing. The more straightforward the data, the better the efficiency.
3. `SELECT` results depend strongly on the consistency of the history of object changes. Be accurate when preparing data for inserting. You can get unpredictable results with inconsistent data, such as negative values for non-negative metrics like session depth.

## Algorithm

When ClickHouse merges data parts, it deletes each pair of rows that have the same primary key and version and different `Sign`. The order of rows does not matter.

When ClickHouse inserts data, it orders rows by the primary key. If the `Version` column is not in the primary key, ClickHouse adds it to the primary key implicitly as the last field and uses it for ordering.

## Selecting Data

ClickHouse does not guarantee that all of the rows with the same primary key will be in the same resulting data part or even on the same physical server. This is true both for writing the data and for subsequent merging of the data parts. In addition, ClickHouse processes `SELECT` queries with multiple threads, and it cannot predict the order of rows in the result. This means that aggregation is required if there is a need to get completely “collapsed” data from a `VersionedCollapsingMergeTree` table.

To finalize collapsing, write a query with a `GROUP BY` clause and aggregate functions that account for the sign. For example, to calculate quantity, use `sum(Sign)` instead of `count()`. To calculate the sum of something, use `sum(Sign * x)` instead of `sum(x)`, and add `HAVING sum(Sign) > 0`.

The aggregates `count`, `sum` and `avg` can be calculated this way. The aggregate `uniq` can be calculated if an object has at least one non-collapsed state. The aggregates `min` and `max` can't be calculated because `VersionedCollapsingMergeTree` does not save the history of values of collapsed states.

If you need to extract the data with “collapsing” but without aggregation (for example, to check whether rows are present whose newest values match certain conditions), you can use the `FINAL` modifier for the `FROM` clause. This approach is inefficient and should not be used with large tables.

## Example of Use

Example data:

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

Creating the table:

```

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

```

Inserting the data:

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

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

We use two `INSERT` queries to create two different data parts. If we insert the data with a single query, ClickHouse creates one data part and will never perform any merge.

Getting the data:

```
SELECT * FROM UAct
```

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

What do we see here and where are the collapsed parts?

We created two data parts using two `INSERT` queries. The `SELECT` query was performed in two threads, and the result is a random order of rows.

Collapsing did not occur because the data parts have not been merged yet. ClickHouse merges data parts at an unknown point in time which we cannot predict.

This is why we need aggregation:

```

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

If we do not need aggregation and want to force collapsing, we can use the `FINAL` modifier for the `FROM` clause.

```
SELECT * FROM UAct FINAL
```

UserID	PageViews	Duration	Sign	Version
4324182021466249494	6	185	1	2

This is a very inefficient way to select data. Don't use it for large tables.

## GraphiteMergeTree

This engine is designed for thinning and aggregating/averaging (rollup) [Graphite](#) data. It may be helpful to developers who want to use ClickHouse as a data store for Graphite.

You can use any ClickHouse table engine to store the Graphite data if you do not need rollup, but if you need a rollup use `GraphiteMergeTree`. The engine reduces the volume of storage and increases the efficiency of queries from Graphite.

The engine inherits properties from [MergeTree](#).

## Creating a Table

```
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, ...]
```

See a detailed description of the [CREATE TABLE](#) query.

A table for the Graphite data should have the following columns for the following data:

- Metric name (Graphite sensor). Data type: `String`.
- Time of measuring the metric. Data type: `DateTime`.
- Value of the metric. Data type: any numeric.
- Version of the metric. Data type: any numeric (ClickHouse saves the rows with the highest version or the last written if versions are the same. Other rows are deleted during the merge of data parts).

The names of these columns should be set in the rollup configuration.

### GraphiteMergeTree parameters

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

### Query clauses

When creating a `GraphiteMergeTree` table, the same [clauses](#) are required, as when creating a `MergeTree` table.

► [Deprecated Method for Creating a Table](#)

# Rollup Configuration

The settings for rollup are defined by the `graphite_rollup` parameter in the server configuration. The name of the parameter could be any. You can create several configurations and use them for different tables.

Rollup configuration structure:

```
required-columns
patterns
```

## Required Columns

- `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`. Default value: `Value`.
- `version_column_name` — The name of the column storing the version of the metric. Default value: `Timestamp`.

## Patterns

Structure of the `patterns` section:

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

## Attention

Patterns must be strictly ordered:

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

When processing a row, ClickHouse checks the rules in the `pattern` sections. Each of `pattern` (including `default`) sections can contain `function` parameter for aggregation, `retention` parameters or both. If the metric name matches the `regexp`, the rules from the `pattern` section (or sections) are applied; otherwise, the rules from the `default` section are used.

Fields for `pattern` and `default` sections:

- `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`]. Accepted functions: `min` / `max` / `any` / `avg`. The average is calculated imprecisely, like the average of the averages.

## Configuration Example

```
<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>
    </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>
```

## Warning

Data rollup is performed during merges. Usually, for old partitions, merges are not started, so for rollup it is necessary to trigger an unscheduled merge using **optimize**. Or use additional tools, for example **graphite-ch-optimizer**.

## Log Engine Family

These engines were developed for scenarios when you need to quickly write many small tables (up to about 1 million rows) and read them later as a whole.

Engines of the family:

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

[Log](#) family table engines can store data to [HDFS](#) or [S3](#) distributed file systems.

## Common Properties

Engines:

- Store data on a disk.
- Append data to the end of file when writing.
- Support locks for concurrent data access.

During `INSERT` queries, the table is locked, and other queries for reading and writing data both wait for the table to unlock. If there are no data writing queries, any number of data reading queries can be performed concurrently.

- Do not support [mutations](#).
- Do not support indexes.

This means that `SELECT` queries for ranges of data are not efficient.

- Do not write data atomically.

You can get a table with corrupted data if something breaks the write operation, for example, abnormal server shutdown.

## Differences

The `TinyLog` engine is the simplest in the family and provides the poorest functionality and lowest efficiency. The `TinyLog` engine does not support parallel data reading by several threads in a single query. It reads data slower than other engines in the family that support parallel reading from a single query and it uses almost as many file descriptors as the `Log` engine because it stores each column in a separate file. Use it only in simple scenarios.

The `Log` and `StripeLog` engines support parallel data reading. When reading data, ClickHouse uses multiple threads. Each thread processes a separate data block. The `Log` engine uses a separate file for each column of the table. `StripeLog` stores all the data in one file. As a result, the `StripeLog` engine uses fewer file descriptors, but the `Log` engine provides higher efficiency when reading data.

## Stripelog

This engine belongs to the family of log engines. See the common properties of log engines and their differences in the [Log Engine Family](#) article.

Use this engine in scenarios when you need to write many tables with a small amount of data (less than 1 million rows).

## Creating a Table

```
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
```

See the detailed description of the [CREATE TABLE](#) query.

## Writing the Data

The StripeLog engine stores all the columns in one file. For each `INSERT` query, ClickHouse appends the data block to the end of a table file, writing columns one by one.

For each table ClickHouse writes the files:

- `data.bin` — Data file.
- `index.mrk` — File with marks. Marks contain offsets for each column of each data block inserted.

The StripeLog engine does not support the `ALTER UPDATE` and `ALTER DELETE` operations.

## Reading the Data

The file with marks allows ClickHouse to parallelize the reading of data. This means that a `SELECT` query returns rows in an unpredictable order. Use the `ORDER BY` clause to sort rows.

## Example of Use

Creating a table:

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

Inserting data:

```
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')
```

We used two `INSERT` queries to create two data blocks inside the `data.bin` file.

ClickHouse uses multiple threads when selecting data. Each thread reads a separate data block and returns resulting rows independently as it finishes. As a result, the order of blocks of rows in the output does not match the order of the same blocks in the input in most cases. For example:

```
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

Sorting the results (ascending order by default):

```
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

## Log

The engine belongs to the family of Log engines. See the common properties of Log engines and their differences in the [Log Engine Family](#) article.

Log differs from [TinyLog](#) in that a small file of "marks" resides with the column files. These marks are written on every data block and contain offsets that indicate where to start reading the file in order to skip the specified number of rows. This makes it possible to read table data in multiple threads.

For concurrent data access, the read operations can be performed simultaneously, while write operations block reads and each other.

The Log engine does not support indexes. Similarly, if writing to a table failed, the table is broken, and reading from it returns an error. The Log engine is appropriate for temporary data, write-once tables, and for testing or demonstration purposes.

## TinyLog

The engine belongs to the log engine family. See [Log Engine Family](#) for common properties of log engines and their differences.

This table engine is typically used with the write-once method: write data one time, then read it as many times as necessary. For example, you can use TinyLog-type tables for intermediary data that is processed in small batches. Note that storing data in a large number of small tables is inefficient.

Queries are executed in a single stream. In other words, this engine is intended for relatively small tables (up to about 1,000,000 rows). It makes sense to use this table engine if you have many small tables, since it's simpler than the Log engine (fewer files need to be opened).

## Special Table Engines

There are three main categories of table engines:

- [MergeTree engine family](#) for main production use.
- [Log engine family](#) for small temporary data.
- [Table engines for integrations](#).

The remaining engines are unique in their purpose and are not grouped into families yet, thus they are placed in this “special” category.

## Distributed Table Engine

Tables with Distributed engine do not store any data of their own, but allow distributed query processing on multiple servers.

Reading is automatically parallelized. During a read, the table indexes on remote servers are used, if there are any.

The Distributed engine accepts parameters:

- the cluster name in the server's config file
- the name of a remote database
- the name of a remote table
- (optionally) sharding key
- (optionally) policy name, it will be used to store temporary files for async send

See also:

- [insert\\_distributed\\_sync](#) setting
- [MergeTree](#) for the examples

Also, it accepts the following settings:

- `fsync_after_insert` - do the `fsync` for the file data after asynchronous insert to Distributed. Guarantees that the OS flushed the whole inserted data to a file **on the initiator node** disk.
- `fsync_directories` - do the `fsync` for directories. Guarantees that the OS refreshed directory metadata after operations related to asynchronous inserts on Distributed table (after insert, after sending the data to shard, etc).
- `bytes_to_throw_insert` - if more than this number of compressed bytes will be pending for async INSERT, an exception will be thrown. 0 - do not throw. Default 0.
- `bytes_to_delay_insert` - if more than this number of compressed bytes will be pending for async INSERT, the query will be delayed. 0 - do not delay. Default 0.
- `max_delay_to_insert` - max delay of inserting data into Distributed table in seconds, if there are a lot of pending bytes for async send. Default 60.
- `monitor_batch_inserts` - same as [distributed\\_directory\\_monitor\\_batch\\_inserts](#)
- `monitor_split_batch_on_failure` - same as [distributed\\_directory\\_monitor\\_split\\_batch\\_on\\_failure](#)
- `monitor_sleep_time_ms` - same as [distributed\\_directory\\_monitor\\_sleep\\_time\\_ms](#)
- `monitor_max_sleep_time_ms` - same as [distributed\\_directory\\_monitor\\_max\\_sleep\\_time\\_ms](#)

### Note

#### Durability settings (`fsync_...`):

- Affect only asynchronous INSERTs (i.e. `insert_distributed_sync=false`) when data first stored on the initiator node disk and later asynchronously send to shards.

- May significantly decrease the inserts' performance
- Affect writing the data stored inside Distributed table folder into the **node which accepted your insert**. If you need to have guarantees of writing data to underlying MergeTree tables - see durability settings (...fsync...) in system.merge\_tree\_settings

For **Insert limit settings** (...\_insert) see also:

- **insert\_distributed\_sync** setting
- **prefer\_localhost\_replica** setting
- `bytes_to_throw_insert` handled before `bytes_to_delay_insert`, so you should not set it to the value less than `bytes_to_delay_insert`

Example:

```
Distributed(logs, default, hits[, sharding_key[, policy_name]])
SETTINGS
    fsync_after_insert=0,
    fsync_directories=0;
```

Data will be read from all servers in the `logs` cluster, from the `default.hits` table located on every server in the cluster.

Data is not only read but is partially processed on the remote servers (to the extent that this is possible). For example, for a query with GROUP BY, data will be aggregated on remote servers, and the intermediate states of aggregate functions will be sent to the requestor server. Then data will be further aggregated.

Instead of the database name, you can use a constant expression that returns a string. For example: `currentDatabase()`.

`logs` – The cluster name in the server's config file.

Clusters are set like this:

```

<remote_servers>
  <logs>
    <!-- Inter-server per-cluster secret for Distributed queries
        default: no secret (no authentication will be performed)

        If set, then Distributed queries will be validated on shards, so at least:
        - such cluster should exist on the shard,
        - such cluster should have the same secret.

        And also (and which is more important), the initial_user will
        be used as current user for the query.
    -->
    <!-- <secret></secret> -->
  <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>
      <!-- Optional. Priority of the replica for load balancing (see also load_balancing setting). Default: 1 (less
          value has more priority). -->
      <priority>1</priority>
      <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>

```

Here a cluster is defined with the name `logs` that consists of two shards, each of which contains two replicas.

Shards refer to the servers that contain different parts of the data (in order to read all the data, you must access all the shards).

Replicas are duplicating servers (in order to read all the data, you can access the data on any one of the replicas).

Cluster names must not contain dots.

The parameters `host`, `port`, and optionally `user`, `password`, `secure`, `compression` are specified for each server:

- `host` – The address of the remote server. You can use either the domain or the IPv4 or IPv6 address. If you specify the domain, the server makes a DNS request when it starts, and the result is stored as long as the server is running. If the DNS request fails, the server does not start. If you change the DNS record, restart the server.
- `port` – The TCP port for messenger activity (`tcp_port` in the config, usually set to 9000). Do not confuse it with `http_port`.
- `user` – Name of the user for connecting to a remote server. Default value: `default`. This user must have access to connect to the specified server. Access is configured in the `users.xml` file. For more information, see the section [Access rights](#).
- `password` – The password for connecting to a remote server (not masked). Default value: empty string.

- `secure` - Use ssl for connection, usually you also should define `port = 9440`. Server should listen on `<tcp_port_secure>9440</tcp_port_secure>` and have correct certificates.
- `compression` - Use data compression. Default value: true.

When specifying replicas, one of the available replicas will be selected for each of the shards when reading. You can configure the algorithm for load balancing (the preference for which replica to access) – see the [load\\_balancing](#) setting.

If the connection with the server is not established, there will be an attempt to connect with a short timeout. If the connection failed, the next replica will be selected, and so on for all the replicas. If the connection attempt failed for all the replicas, the attempt will be repeated the same way, several times. This works in favour of resiliency, but does not provide complete fault tolerance: a remote server might accept the connection, but might not work, or work poorly.

You can specify just one of the shards (in this case, query processing should be called remote, rather than distributed) or up to any number of shards. In each shard, you can specify from one to any number of replicas. You can specify a different number of replicas for each shard.

You can specify as many clusters as you wish in the configuration.

To view your clusters, use the `system.clusters` table.

The Distributed engine allows working with a cluster like a local server. However, the cluster is inextensible: you must write its configuration in the server config file (even better, for all the cluster's servers).

The Distributed engine requires writing clusters to the config file. Clusters from the config file are updated on the fly, without restarting the server. If you need to send a query to an unknown set of shards and replicas each time, you do not need to create a Distributed table – use the `remote` table function instead. See the section [Table functions](#).

There are two methods for writing data to a cluster:

First, you can define which servers to write which data to and perform the write directly on each shard. In other words, perform `INSERT` in the tables that the distributed table “looks at”. This is the most flexible solution as you can use any sharding scheme, which could be non-trivial due to the requirements of the subject area. This is also the most optimal solution since data can be written to different shards completely independently.

Second, you can perform `INSERT` in a Distributed table. In this case, the table will distribute the inserted data across the servers itself. In order to write to a Distributed table, it must have a sharding key set (the last parameter). In addition, if there is only one shard, the write operation works without specifying the sharding key, since it does not mean anything in this case.

Each shard can have a weight defined in the config file. By default, the weight is equal to one. Data is distributed across shards in the amount proportional to the shard weight. For example, if there are two shards and the first has a weight of 9 while the second has a weight of 10, the first will be sent 9 / 19 parts of the rows, and the second will be sent 10 / 19.

Each shard can have the `internal_replication` parameter defined in the config file.

If this parameter is set to `true`, the write operation selects the first healthy replica and writes data to it. Use this alternative if the Distributed table “looks at” replicated tables. In other words, if the table where data will be written is going to replicate them itself.

If it is set to `false` (the default), data is written to all replicas. In essence, this means that the Distributed table replicates data itself. This is worse than using replicated tables, because the consistency of replicas is not checked, and over time they will contain slightly different data.

To select the shard that a row of data is sent to, the sharding expression is analyzed, and its remainder is taken from dividing it by the total weight of the shards. The row is sent to the shard that corresponds to the half-interval of the remainders from `prev_weights` to `prev_weights + weight`, where `prev_weights` is the total weight of the shards with the smallest number, and `weight` is the weight of this shard. For example, if there are two shards, and the first has a weight of 9 while the second has a weight of 10, the row will be sent to the first shard for the remainders from the range [0, 9), and to the second for the remainders from the range [9, 19).

The sharding expression can be any expression from constants and table columns that returns an integer. For example, you can use the expression `rand()` for random distribution of data, or `UserID` for distribution by the remainder from dividing the user's ID (then the data of a single user will reside on a single shard, which simplifies running IN and JOIN by users). If one of the columns is not distributed evenly enough, you can wrap it in a hash function: `intHash64(UserID)`.

A simple remainder from the division is a limited solution for sharding and isn't always appropriate. It works for medium and large volumes of data (dozens of servers), but not for very large volumes of data (hundreds of servers or more). In the latter case, use the sharding scheme required by the subject area, rather than using entries in Distributed tables.

SELECT queries are sent to all the shards and work regardless of how data is distributed across the shards (they can be distributed completely randomly). When you add a new shard, you do not have to transfer old data into it. Instead, you can write new data to it by using a heavier weight – the data will be distributed slightly unevenly, but queries will work correctly and efficiently.

You should be concerned about the sharding scheme in the following cases:

- Queries are used that require joining data (IN or JOIN) by a specific key. If data is sharded by this key, you can use local IN or JOIN instead of GLOBAL IN or GLOBAL JOIN, which is much more efficient.
- A large number of servers is used (hundreds or more) with a large number of small queries (queries of individual clients - websites, advertisers, or partners). In order for the small queries to not affect the entire cluster, it makes sense to locate data for a single client on a single shard. Alternatively, as we've done in Yandex.Metrica, you can set up bi-level sharding: divide the entire cluster into "layers", where a layer may consist of multiple shards. Data for a single client is located on a single layer, but shards can be added to a layer as necessary, and data is randomly distributed within them. Distributed tables are created for each layer, and a single shared distributed table is created for global queries.

Data is written asynchronously. When inserted in the table, the data block is just written to the local file system. The data is sent to the remote servers in the background as soon as possible. The periodicity for sending data is managed by the `distributed_directory_monitor_sleep_time_ms` and `distributed_directory_monitor_max_sleep_time_ms` settings. The Distributed engine sends each file with inserted data separately, but you can enable batch sending of files with the `distributed_directory_monitor_batch_inserts` setting. This setting improves cluster performance by better utilizing local server and network resources. You should check whether data is sent successfully by checking the list of files (data waiting to be sent) in the table directory: `/var/lib/clickhouse/data/database/table/`. The number of threads performing background tasks can be set by `background_distributed_schedule_pool_size` setting.

If the server ceased to exist or had a rough restart (for example, after a device failure) after an INSERT to a Distributed table, the inserted data might be lost. If a damaged data part is detected in the table directory, it is transferred to the `broken` subdirectory and no longer used.

When the `max_parallel_replicas` option is enabled, query processing is parallelized across all replicas within a single shard. For more information, see the section [max\\_parallel\\_replicas](#).

## Virtual Columns

- `_shard_num` — Contains the `shard_num` value from the table `system.clusters`. Type: `UInt32`.

## Note

Since `remote` and `cluster` table functions internally create temporary Distributed table, `_shard_num` is available there too.

## See Also

- [Virtual columns](#) description
- [background\\_distributed\\_schedule\\_pool\\_size](#) setting
- [shardNum\(\)](#) and [shardCount\(\)](#) functions

# Dictionary Table Engine

The Dictionary engine displays the [dictionary](#) data as a ClickHouse table.

## Example

As an example, consider a dictionary of `products` with the following configuration:

```
<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>
```

Query the dictionary data:

```
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

You can use the [dictGet\\*](#) function to get the dictionary data in this format.

This view isn't helpful when you need to get raw data, or when performing a `JOIN` operation. For these cases, you can use the `Dictionary` engine, which displays the dictionary data in a table.

Syntax:

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

Usage example:

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

```
Ok
```

Take a look at what's in the table.

```
select * from products limit 1;
```

product_id	title
152689	Some item

## See Also

- [Dictionary function](#)

# Merge Table Engine

The `Merge` engine (not to be confused with `MergeTree`) does not store data itself, but allows reading from any number of other tables simultaneously.

Reading is automatically parallelized. Writing to a table is not supported. When reading, the indexes of tables that are actually being read are used, if they exist.

The `Merge` engine accepts parameters: the database name and a regular expression for tables.

# Examples

Example 1:

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

Data will be read from the tables in the `hits` database that have names that match the regular expression `'^WatchLog'`.

Instead of the database name, you can use a constant expression that returns a string. For example, `currentDatabase()`.

Regular expressions — [re2](#) (supports a subset of PCRE), case-sensitive.

See the notes about escaping symbols in regular expressions in the “match” section.

When selecting tables to read, the `Merge` table itself will not be selected, even if it matches the regex. This is to avoid loops.

It is possible to create two `Merge` tables that will endlessly try to read each others’ data, but this is not a good idea.

The typical way to use the `Merge` engine is for working with a large number of `TinyLog` tables as if with a single table.

Example 2:

Let’s say you have a old table (`WatchLog_old`) and decided to change partitioning without moving data to a new table (`WatchLog_new`) and you need to see data from both tables.

```
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

## Virtual Columns

- `_table` — Contains the name of the table from which data was read. Type: [String](#).

You can set the constant conditions on `_table` in the `WHERE/PREWHERE` clause (for example, `WHERE _table='xyz'`). In this case the read operation is performed only for that tables where the condition on `_table` is satisfied, so the `_table` column acts as an index.

## See Also

- [Virtual columns](#)

# File Table Engine

The File table engine keeps the data in a file in one of the supported [file formats](#) (TabSeparated, Native, etc.).

Usage scenarios:

- Data export from ClickHouse to file.
- Convert data from one format to another.
- Updating data in ClickHouse via editing a file on a disk.

## Usage in ClickHouse Server

File(Format)

The `Format` parameter specifies one of the available file formats. To perform `SELECT` queries, the format must be supported for input, and to perform `INSERT` queries – for output. The available formats are listed in the [Formats](#) section.

ClickHouse does not allow specifying filesystem path for `File`. It will use folder defined by `path` setting in server configuration.

When creating table using `File(Format)` it creates empty subdirectory in that folder. When data is written to that table, it's put into `data.Format` file in that subdirectory.

You may manually create this subfolder and file in server filesystem and then [ATTACH](#) it to table information with matching name, so you can query data from that file.

### Warning

Be careful with this functionality, because ClickHouse does not keep track of external changes to such files. The result of simultaneous writes via ClickHouse and outside of ClickHouse is undefined.

## Example

1. Set up the `file_engine_table` table:

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

By default ClickHouse will create folder `/var/lib/clickhouse/data/default/file_engine_table`.

2. Manually create `/var/lib/clickhouse/data/default/file_engine_table/data.TabSeparated` containing:

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

3. Query the data:

```
SELECT * FROM file_engine_table
```

name	value
one	1
two	2

## Usage in ClickHouse-local

In [clickhouse-local](#) File engine accepts file path in addition to Format. Default input/output streams can be specified using numeric or human-readable names like 0 or `stdin`, 1 or `stdout`. It is possible to read and write compressed files based on an additional engine parameter or file extension (`gz`, `br` or `xz`).

### Example:

```
$ 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"
```

## Details of Implementation

- Multiple `SELECT` queries can be performed concurrently, but `INSERT` queries will wait each other.
- Supported creating new file by `INSERT` query.
- If file exists, `INSERT` would append new values in it.
- Not supported:
  - `ALTER`
  - `SELECT ... SAMPLE`
  - Indices
  - Replication

## Null Table Engine

When writing to a `Null` table, data is ignored. When reading from a `Null` table, the response is empty.

### Hint

However, you can create a materialized view on a `Null` table. So the data written to the table will end up affecting the view, but original raw data will still be discarded.

## Set Table Engine

A data set that is always in RAM. It is intended for use on the right side of the `IN` operator (see the section “`IN` operators”).

You can use `INSERT` to insert data in the table. New elements will be added to the data set, while duplicates will be ignored.

But you can't perform `SELECT` from the table. The only way to retrieve data is by using it in the right half of the `IN` operator.

Data is always located in RAM. For `INSERT`, the blocks of inserted data are also written to the directory of tables on the disk. When starting the server, this data is loaded to RAM. In other words, after restarting, the data remains in place.

For a rough server restart, the block of data on the disk might be lost or damaged. In the latter case, you may need to manually delete the file with damaged data.

## Limitations and Settings

When creating a table, the following settings are applied:

- `persistent`

## Join Table Engine

Optional prepared data structure for usage in `JOIN` operations.

### Note

This is not an article about the `JOIN clause` itself.

## Creating a Table

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
(
    name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1] [TTL expr1],
    name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2] [TTL expr2],
) ENGINE = Join(join_strictness, join_type, k1[, k2, ...])
```

See the detailed description of the `CREATE TABLE` query.

### Engine Parameters

- `join_strictness` – `JOIN strictness`.
- `join_type` – `JOIN type`.
- `k1[, k2, ...]` – Key columns from the `USING` clause that the `JOIN` operation is made with.

Enter `join_strictness` and `join_type` parameters without quotes, for example, `Join(ANY, LEFT, col1)`. They must match the `JOIN` operation that the table will be used for. If the parameters do not match, ClickHouse does not throw an exception and may return incorrect data.

## Specifics and Recommendations

### Data Storage

Join table data is always located in the RAM. When inserting rows into a table, ClickHouse writes data blocks to the directory on the disk so that they can be restored when the server restarts.

If the server restarts incorrectly, the data block on the disk might get lost or damaged. In this case, you may need to manually delete the file with damaged data.

### Selecting and Inserting Data

You can use `INSERT` queries to add data to the Join-engine tables. If the table was created with the `ANY` strictness, data for duplicate keys are ignored. With the `ALL` strictness, all rows are added.

Main use-cases for Join-engine tables are following:

- Place the table to the right side in a `JOIN` clause.
- Call the `joinGet` function, which lets you extract data from the table the same way as from a dictionary.

## Deleting Data

`ALTER DELETE` queries for Join-engine tables are implemented as `mutations`. `DELETE` mutation reads filtered data and overwrites data of memory and disk.

## Limitations and Settings

When creating a table, the following settings are applied:

- `join_use_nulls`
- `max_rows_in_join`
- `max_bytes_in_join`
- `join_overflow_mode`
- `join_any_take_last_row`
- `persistent`

The Join-engine tables can't be used in `GLOBAL JOIN` operations.

The Join-engine allows to specify `join_use_nulls` setting in the `CREATE TABLE` statement. `SELECT` query should have the same `join_use_nulls` value.

## Usage Examples

Creating the left-side table:

```
CREATE TABLE id_val(`id` UInt32, `val` UInt32) ENGINE = TinyLog;
```

```
INSERT INTO id_val VALUES (1,11)(2,12)(3,13);
```

Creating the right-side Join table:

```
CREATE TABLE id_val_join(`id` UInt32, `val` UInt8) ENGINE = Join(ANY, LEFT, id);
```

```
INSERT INTO id_val_join VALUES (1,21)(1,22)(3,23);
```

Joining the tables:

```
SELECT * FROM id_val ANY LEFT JOIN id_val_join USING (id);
```

id	val	id_val	join.val
1	11		21
2	12		0
3	13		23

As an alternative, you can retrieve data from the `Join` table, specifying the join key value:

```
SELECT joinGet('id_val_join', 'val', toUInt32(1));
```

```
joinGet('id_val_join', 'val', toUInt32(1))—  
21 |
```

Deleting a row from the `Join` table:

```
ALTER TABLE id_val_join DELETE WHERE id = 3;
```

```
id—val  
1 | 21 |
```

## URL Table Engine

Queries data to/from a remote HTTP/HTTPS server. This engine is similar to the [File](#) engine.

Syntax: `URL(URL, Format)`

### Usage

The format must be one that ClickHouse can use in

`SELECT` queries and, if necessary, in `INSERTs`. For the full list of supported formats, see [Formats](#).

The `URL` must conform to the structure of a Uniform Resource Locator. The specified URL must point to a server

that uses HTTP or HTTPS. This does not require any additional headers for getting a response from the server.

`INSERT` and `SELECT` queries are transformed to `POST` and `GET` requests, respectively. For processing `POST` requests, the remote server must support [Chunked transfer encoding](#).

You can limit the maximum number of HTTP GET redirect hops using the [max\\_http\\_get\\_redirects](#) setting.

### Example

1. Create a `url_engine_table` table on the server :

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

2. Create a basic HTTP server using the standard Python 3 tools and start it:

```

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. Request data:

```
SELECT * FROM url_engine_table
```

word	value
Hello	1
World	2

## Details of Implementation

- Reads and writes can be parallel
- Not supported:
  - ALTER and SELECT...SAMPLE operations.
  - Indexes.
  - Replication.

## View Table Engine

Used for implementing views (for more information, see the [CREATE VIEW query](#)). It does not store data, but only stores the specified `SELECT` query. When reading from a table, it runs this query (and deletes all unnecessary columns from the query).

## MaterializedView Table Engine

Used for implementing materialized views (for more information, see [CREATE VIEW](#)). For storing data, it uses a different engine that was specified when creating the view. When reading from a table, it just uses that engine.

## Memory Table Engine

The Memory engine stores data in RAM, in uncompressed form. Data is stored in exactly the same form as it is received when read. In other words, reading from this table is completely free.

Concurrent data access is synchronized. Locks are short: read and write operations do not block each other.

Indexes are not supported. Reading is parallelized.

Maximal productivity (over 10 GB/sec) is reached on simple queries, because there is no reading from the disk, decompressing, or deserializing data. (We should note that in many cases, the productivity of the MergeTree engine is almost as high.)

When restarting a server, data disappears from the table and the table becomes empty.

Normally, using this table engine is not justified. However, it can be used for tests, and for tasks where maximum speed is required on a relatively small number of rows (up to approximately 100,000,000).

The Memory engine is used by the system for temporary tables with external query data (see the section “External data for processing a query”), and for implementing `GLOBAL IN` (see the section “IN operators”).

## Buffer Table Engine

Buffers the data to write in RAM, periodically flushing it to another table. During the read operation, data is read from the buffer and the other table simultaneously.

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

Engine parameters:

- `database` – Database name. Instead of the database name, you can use a constant expression that returns a string.
- `table` – Table to flush data to.
- `num_layers` – Parallelism layer. Physically, the table will be represented as `num_layers` of independent buffers. Recommended value: 16.
- `min_time`, `max_time`, `min_rows`, `max_rows`, `min_bytes`, and `max_bytes` – Conditions for flushing data from the buffer.

Optional engine parameters:

- `flush_time`, `flush_rows`, `flush_bytes` – Conditions for flushing data from the buffer, that will happen only in background (omitted or zero means no `flush*` parameters).

Data is flushed from the buffer and written to the destination table if all the `min*` conditions or at least one `max*` condition are met.

Also, if at least one `flush*` condition are met flush initiated in background, this is different from `max*`, since `flush*` allows you to configure background flushes separately to avoid adding latency for `INSERT` (into Buffer) queries.

- `min_time`, `max_time`, `flush_time` – Condition for the time in seconds from the moment of the first write to the buffer.
- `min_rows`, `max_rows`, `flush_rows` – Condition for the number of rows in the buffer.
- `min_bytes`, `max_bytes`, `flush_bytes` – Condition for the number of bytes in the buffer.

During the write operation, data is inserted to a `num_layers` number of random buffers. Or, if the data part to insert is large enough (greater than `max_rows` or `max_bytes`), it is written directly to the destination table, omitting the buffer.

The conditions for flushing the data are calculated separately for each of the `num_layers` buffers. For example, if `num_layers = 16` and `max_bytes = 100000000`, the maximum RAM consumption is 1.6 GB.

Example:

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

Creating a `merge.hits_buffer` table with the same structure as `merge.hits` and using the Buffer engine. When writing to this table, data is buffered in RAM and later written to the ‘`merge.hits`’ table. 16 buffers are created. The data in each of them is flushed if either 100 seconds have passed, or one million rows have been written, or 100 MB of data have been written; or if simultaneously 10 seconds have passed and 10,000 rows and 10 MB of data have been written. For example, if just one row has been written, after 100 seconds it will be flushed, no matter what. But if many rows have been written, the data will be flushed sooner.

When the server is stopped, with `DROP TABLE` or `DETACH TABLE`, buffer data is also flushed to the destination table.

You can set empty strings in single quotation marks for the database and table name. This indicates the absence of a destination table. In this case, when the data flush conditions are reached, the buffer is simply cleared. This may be useful for keeping a window of data in memory.

When reading from a Buffer table, data is processed both from the buffer and from the destination table (if there is one).

Note that the Buffer tables does not support an index. In other words, data in the buffer is fully scanned, which might be slow for large buffers. (For data in a subordinate table, the index that it supports will be used.)

If the set of columns in the Buffer table does not match the set of columns in a subordinate table, a subset of columns that exist in both tables is inserted.

If the types do not match for one of the columns in the Buffer table and a subordinate table, an error message is entered in the server log, and the buffer is cleared.

The same thing happens if the subordinate table does not exist when the buffer is flushed.

If you need to run `ALTER` for a subordinate table, and the Buffer table, we recommend first deleting the Buffer table, running `ALTER` for the subordinate table, then creating the Buffer table again.

## Attention

Running `ALTER` on the Buffer table in releases made before 28 Sep 2020 will cause a `Block structure mismatch` error (see [#15117](#)), so deleting the Buffer table and then recreating is the only option. It is advisable to check that this error is fixed in your release before trying to run `ALTER` on the Buffer table.

If the server is restarted abnormally, the data in the buffer is lost.

`FINAL` and `SAMPLE` do not work correctly for Buffer tables. These conditions are passed to the destination table, but are not used for processing data in the buffer. If these features are required we recommend only using the Buffer table for writing, while reading from the destination table.

When adding data to a Buffer, one of the buffers is locked. This causes delays if a read operation is simultaneously being performed from the table.

Data that is inserted to a Buffer table may end up in the subordinate table in a different order and in different blocks. Because of this, a Buffer table is difficult to use for writing to a CollapsingMergeTree correctly. To avoid problems, you can set `num_layers` to 1.

If the destination table is replicated, some expected characteristics of replicated tables are lost when writing to a Buffer table. The random changes to the order of rows and sizes of data parts cause data deduplication to quit working, which means it is not possible to have a reliable ‘exactly once’ write to replicated tables.

Due to these disadvantages, we can only recommend using a Buffer table in rare cases.

A Buffer table is used when too many INSERTs are received from a large number of servers over a unit of time and data can’t be buffered before insertion, which means the INSERTs can’t run fast enough.

Note that it does not make sense to insert data one row at a time, even for Buffer tables. This will only produce a speed of a few thousand rows per second, while inserting larger blocks of data can produce over a million rows per second (see the section “Performance”).

## External Data for Query Processing

ClickHouse allows sending a server the data that is needed for processing a query, together with a `SELECT` query. This data is put in a temporary table (see the section “Temporary tables”) and can be used in the query (for example, in `IN` operators).

For example, if you have a text file with important user identifiers, you can upload it to the server along with a query that uses filtration by this list.

If you need to run more than one query with a large volume of external data, do not use this feature. It is better to upload the data to the DB ahead of time.

External data can be uploaded using the command-line client (in non-interactive mode), or using the HTTP interface.

In the command-line client, you can specify a parameters section in the format

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

You may have multiple sections like this, for the number of tables being transmitted.

**-external** – Marks the beginning of a clause.

**-file** – Path to the file with the table dump, or `-`, which refers to `stdin`.

Only a single table can be retrieved from `stdin`.

The following parameters are optional: **-name** – Name of the table. If omitted, `_data` is used.

**-format** – Data format in the file. If omitted, `TabSeparated` is used.

One of the following parameters is required:**-types** – A list of comma-separated column types. For example: `UInt64, String`. The columns will be named `_1, _2, ...`

**-structure** – The table structure in the format `userID UInt64, URL String`. Defines the column names and types.

The files specified in ‘file’ will be parsed by the format specified in ‘format’, using the data types specified in ‘types’ or ‘structure’. The table will be uploaded to the server and accessible there as a temporary table with the name in ‘name’.

Examples:

```
$ 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
```

When using the HTTP interface, external data is passed in the multipart/form-data format. Each table is transmitted as a separate file. The table name is taken from the file name. The `query_string` is passed the parameters `name_format`, `name_types`, and `name_structure`, where `name` is the name of the table that these parameters correspond to. The meaning of the parameters is the same as when using the command-line client.

Example:

```
$ 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=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
```

For distributed query processing, the temporary tables are sent to all the remote servers.

## GenerateRandom Table Engine

The `GenerateRandom` table engine produces random data for given table schema.

Usage examples:

- Use in test to populate reproducible large table.
- Generate random input for fuzzing tests.

## Usage in ClickHouse Server

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

The `max_array_length` and `max_string_length` parameters specify maximum length of all array columns and strings correspondingly in generated data.

Generate table engine supports only `SELECT` queries.

It supports all `DataTypes` that can be stored in a table except `LowCardinality` and `AggregateFunction`.

## Example

1. Set up the `generate_engine_table` table:

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

## 2. Query the data:

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

name	value
c4xj	1412771199
r	1791099446
7#\$	124312908

## Details of Implementation

- Not supported:
  - ALTER
  - SELECT ... SAMPLE
  - INSERT
  - Indices
  - Replication

## Database Engines

Database engines allow you to work with tables.

By default, ClickHouse uses database engine [Atomic](#). It provides configurable [table engines](#) and an [SQL dialect](#).

You can also use the following database engines:

- [MySQL](#)
- [MaterializedMySQL](#)
- [Lazy](#)
- [Atomic](#)
- [PostgreSQL](#)
- [Replicated](#)
- [SQLite](#)

## [experimental] MaterializedMySQL

### Warning

This is an experimental feature that should not be used in production.

Creates ClickHouse database with all the tables existing in MySQL, and all the data in those tables.

ClickHouse server works as MySQL replica. It reads binlog and performs DDL and DML queries.

## Creating a Database

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

### Engine Parameters

- `host:port` — MySQL server endpoint.
- `database` — MySQL database name.
- `user` — MySQL user.
- `password` — User password.

### Engine Settings

- `max_rows_in_buffer` — Maximum number of rows that data is allowed to cache in memory (for single table and the cache data unable to query). When this number is exceeded, the data will be materialized. Default: `65 505`.
- `max_bytes_in_buffer` — Maximum number of bytes that data is allowed to cache in memory (for single table and the cache data unable to query). When this number is exceeded, the data will be materialized. Default: `1 048 576`.
- `max_rows_in_buffers` — Maximum number of rows that data is allowed to cache in memory (for database and the cache data unable to query). When this number is exceeded, the data will be materialized. Default: `65 505`.
- `max_bytes_in_buffers` — Maximum number of bytes that data is allowed to cache in memory (for database and the cache data unable to query). When this number is exceeded, the data will be materialized. Default: `1 048 576`.
- `max_flush_data_time` — Maximum number of milliseconds that data is allowed to cache in memory (for database and the cache data unable to query). When this time is exceeded, the data will be materialized. Default: `1000`.
- `max_wait_time_when_mysql_unavailable` — Retry interval when MySQL is not available (milliseconds). Negative value disables retry. Default: `1000`.
- `allows_query_when_mysql_lost` — Allows to query a materialized table when MySQL is lost. Default: `0 (false)`.

```
CREATE DATABASE mysql ENGINE = MaterializedMySQL('localhost:3306', 'db', 'user', '***')
SETTINGS
    allows_query_when_mysql_lost=true,
    max_wait_time_when_mysql_unavailable=10000;
```

### Settings on MySQL-server Side

For the correct work of `MaterializedMySQL`, there are few mandatory MySQL-side configuration settings that must be set:

- `default_authentication_plugin = mysql_native_password` since `MaterializedMySQL` can only authorize with this method.
- `gtid_mode = on` since GTID based logging is a mandatory for providing correct `MaterializedMySQL` replication.

## Attention

While turning on `gtid_mode` you should also specify `enforce_gtid_consistency = on`.

## Virtual Columns

When working with the `MaterializedMySQL` database engine, `ReplacingMergeTree` tables are used with virtual `_sign` and `_version` columns.

- `_version` — Transaction counter. Type `UInt64`.
- `_sign` — Deletion mark. Type `Int8`. Possible values:
  - `1` — Row is not deleted,
  - `-1` — Row is deleted.

## Data Types Support

MySQL	ClickHouse
TINY	<code>Int8</code>
SHORT	<code>Int16</code>
INT24	<code>Int32</code>
LONG	<code>UInt32</code>
LONGLONG	<code>UInt64</code>
FLOAT	<code>Float32</code>
DOUBLE	<code>Float64</code>
DECIMAL, NEWDECIMAL	<code>Decimal</code>
DATE, NEWDATE	<code>Date</code>
DATETIME, TIMESTAMP	<code>DateTime</code>
DATETIME2, TIMESTAMP2	<code>DateTime64</code>
ENUM	<code>Enum</code>
STRING	<code>String</code>
VARCHAR, VAR_STRING	<code>String</code>
BLOB	<code>String</code>
BINARY	<code>FixedString</code>

`Nullable` is supported.

Other types are not supported. If MySQL table contains a column of such type, ClickHouse throws exception "Unhandled data type" and stops replication.

## Specifics and Recommendations

### Compatibility Restrictions

Apart of the data types limitations there are few restrictions comparing to MySQL databases, that should be resolved before replication will be possible:

- Each table in MySQL should contain PRIMARY KEY.
- Replication for tables, those are containing rows with ENUM field values out of range (specified in ENUM signature) will not work.

### DDL Queries

MySQL DDL queries are converted into the corresponding ClickHouse DDL queries (ALTER, CREATE, DROP, RENAME). If ClickHouse cannot parse some DDL query, the query is ignored.

### Data Replication

MaterializedMySQL does not support direct INSERT, DELETE and UPDATE queries. However, they are supported in terms of data replication:

- MySQL INSERT query is converted into INSERT with \_sign=1.
- MySQL DELETE query is converted into INSERT with \_sign=-1.
- MySQL UPDATE query is converted into INSERT with \_sign=-1 and INSERT with \_sign=1.

### Selecting from MaterializedMySQL Tables

SELECT query from MaterializedMySQL tables has some specifics:

- If \_version is not specified in the SELECT query, FINAL modifier is used. So only rows with MAX(\_version) are selected.
- If \_sign is not specified in the SELECT query, WHERE \_sign=1 is used by default. So the deleted rows are not included into the result set.
- The result includes columns comments in case they exist in MySQL database tables.

### Index Conversion

MySQL PRIMARY KEY and INDEX clauses are converted into ORDER BY tuples in ClickHouse tables.

ClickHouse has only one physical order, which is determined by ORDER BY clause. To create a new physical order, use materialized views.

### Notes

- Rows with \_sign=-1 are not deleted physically from the tables.
- Cascade UPDATE/DELETE queries are not supported by the MaterializedMySQL engine.
- Replication can be easily broken.
- Manual operations on database and tables are forbidden.
- MaterializedMySQL is influenced by optimize\_on\_insert setting. The data is merged in the corresponding table in the MaterializedMySQL database when a table in the MySQL server changes.

# Examples of Use

Queries in MySQL:

```
mysql> CREATE DATABASE db;
mysql> CREATE TABLE db.test (a INT PRIMARY KEY, b INT);
mysql> INSERT INTO db.test VALUES (1, 11), (2, 22);
mysql> DELETE FROM db.test WHERE a=1;
mysql> ALTER TABLE db.test ADD COLUMN c VARCHAR(16);
mysql> UPDATE db.test SET c='Wow!', b=222;
mysql> SELECT * FROM test;
```

a	b	c
2	222	Wow!

Database in ClickHouse, exchanging data with the MySQL server:

The database and the table created:

```
CREATE DATABASE mysql ENGINE = MaterializedMySQL('localhost:3306', 'db', 'user', '***');
SHOW TABLES FROM mysql;
```

name
test

After inserting data:

```
SELECT * FROM mysql.test;
```

a	b
1	11
2	22

After deleting data, adding the column and updating:

```
SELECT * FROM mysql.test;
```

a	b	c
2	222	Wow!

## [experimental] MaterializedPostgreSQL

Creates ClickHouse database with an initial data dump of PostgreSQL database tables and starts replication process, i.e. executes background job to apply new changes as they happen on PostgreSQL database tables in the remote PostgreSQL database.

ClickHouse server works as PostgreSQL replica. It reads WAL and performs DML queries. DDL is not replicated, but can be handled (described below).

# Creating a Database

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

## Engine Parameters

- `host:port` — PostgreSQL server endpoint.
- `database` — PostgreSQL database name.
- `user` — PostgreSQL user.
- `password` — User password.

## Dynamically adding new tables to replication

```
ATTACH TABLE postgres_database.new_table;
```

It will work as well if there is a setting `materialized_postgresql_tables_list`.

## Dynamically removing tables from replication

```
DETACH TABLE postgres_database.table_to_remove;
```

## Settings

- `materialized_postgresql_max_block_size`
- `materialized_postgresql_tables_list`
- `materialized_postgresql_allow_automatic_update`
- `materialized_postgresql_replication_slot`
- `materialized_postgresql_snapshot`

```
CREATE DATABASE database1
ENGINE = MaterializedPostgreSQL('postgres1:5432', 'postgres_database', 'postgres_user', 'postgres_password')
SETTINGS materialized_postgresql_max_block_size = 65536,
         materialized_postgresql_tables_list = 'table1,table2,table3';

SELECT * FROM database1.table1;
```

It is also possible to change settings at run time.

```
ALTER DATABASE postgres_database MODIFY SETTING materialized_postgresql_max_block_size = <new_size>;
```

## Requirements

1. The `wal_level` setting must have a value `logical` and `max_replication_slots` parameter must have a value at least 2 in the PostgreSQL config file.
2. Each replicated table must have one of the following `replica identity`:
  - primary key (by default)

- index

```
postgres# CREATE TABLE postgres_table (a Integer NOT NULL, b Integer, c Integer NOT NULL, d Integer, e Integer NOT NULL);
postgres# CREATE unique INDEX postgres_table_index on postgres_table(a, c, e);
postgres# ALTER TABLE postgres_table REPLICA IDENTITY USING INDEX postgres_table_index;
```

The primary key is always checked first. If it is absent, then the index, defined as replica identity index, is checked.

If the index is used as a replica identity, there has to be only one such index in a table.

You can check what type is used for a specific table with the following command:

```
postgres# SELECT CASE relreplident
  WHEN 'd' THEN 'default'
  WHEN 'n' THEN 'nothing'
  WHEN 'f' THEN 'full'
  WHEN 'i' THEN 'index'
END AS replica_identity
FROM pg_class
WHERE oid = 'postgres_table'::regclass;
```

## Warning

Replication of **TOAST** values is not supported. The default value for the data type will be used.

## Example of Use

```
CREATE DATABASE postgresql_db
ENGINE = MaterializedPostgreSQL('postgres1:5432', 'postgres_database', 'postgres_user', 'postgres_password');

SELECT * FROM postgresql_db.postgres_table;
```

## Notes

### Failover of the logical replication slot

Logical Replication Slots which exist on the primary are not available on standby replicas.

So if there is a failover, new primary (the old physical standby) won't be aware of any slots which were existing with old primary. This will lead to a broken replication from PostgreSQL.

A solution to this is to manage replication slots yourself and define a permanent replication slot (some information can be found [here](#)). You'll need to pass slot name via **materialized\_postgresql\_replication\_slot** setting, and it has to be exported with **EXPORT SNAPSHOT** option. The snapshot identifier needs to be passed via **materialized\_postgresql\_snapshot** setting.

Please note that this should be used only if it is actually needed. If there is no real need for that or full understanding why, then it is better to allow the table engine to create and manage its own replication slot.

### Example (from [@bchrobot](#))

1. Configure replication slot in PostgreSQL.

```
apiVersion: "acid.zalan.do/v1"
kind: postgresql
metadata:
  name: acid-demo-cluster
spec:
  numberOfInstances: 2
  postgresql:
    parameters:
      wal_level: logical
  patroni:
    slots:
      clickhouse_sync:
        type: logical
        database: demodb
        plugin: pgoutput
```

2. Wait for replication slot to be ready, then begin a transaction and export the transaction snapshot identifier:

```
BEGIN;
SELECT pg_export_snapshot();
```

3. In ClickHouse create database:

```
CREATE DATABASE demodb
ENGINE = MaterializedPostgreSQL('postgres1:5432', 'postgres_database', 'postgres_user', 'postgres_password')
SETTINGS
  materialized_postgresql_replication_slot = 'clickhouse_sync',
  materialized_postgresql_snapshot = '0000000A-0000023F-3',
  materialized_postgresql_tables_list = 'table1,table2,table3';
```

4. End the PostgreSQL transaction once replication to ClickHouse DB is confirmed. Verify that replication continues after failover:

```
kubectl exec acid-demo-cluster-0 -c postgres -- su postgres -c 'patronictl failover --candidate acid-demo-cluster-1 --force'
```

## MySQL

Allows to connect to databases on a remote MySQL server and perform `INSERT` and `SELECT` queries to exchange data between ClickHouse and MySQL.

The `MySQL` database engine translate queries to the MySQL server so you can perform operations such as `SHOW TABLES` or `SHOW CREATE TABLE`.

You cannot perform the following queries:

- `RENAME`
- `CREATE TABLE`
- `ALTER`

## Creating a Database

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

### Engine Parameters

- `host:port` — MySQL server address.
- `database` — Remote database name.
- `user` — MySQL user.
- `password` — User password.

## Data Types Support

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	Date
DATETIME, TIMESTAMP	DateTime
BINARY	FixedString

All other MySQL data types are converted into [String](#).

[Nullable](#) is supported.

## Global Variables Support

For better compatibility you may address global variables in MySQL style, as `@@identifier`.

These variables are supported:

- `version`
- `max_allowed_packet`

### Warning

By now these variables are stubs and don't correspond to anything.

Example:

```
SELECT @@version;
```

## Examples of Use

Table in MySQL:

```
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)
```

Database in ClickHouse, exchanging data with the MySQL server:

```
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

## Lazy

Keeps tables in RAM only `expiration_time_in_seconds` seconds after last access. Can be used only with `*Log` tables.

It's optimized for storing many small `*Log` tables, for which there is a long time interval between accesses.

## Creating a Database

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

## Atomic

It supports non-blocking `DROP TABLE` and `RENAME TABLE` queries and atomic `EXCHANGE TABLES` queries. Atomic database engine is used by default.

## Creating a Database

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

## Specifics and recommendations

### Table UUID

All tables in database `Atomic` have persistent `UUID` and store data in directory `/clickhouse_path/store/xxx/xxxxxxxx-xxxx-yyyy-yyyy-yyyyyyyyyyyy/`, where `xxxxxxxx-xxxx-yyyy-yyyy-yyyyyyyyyyyy` is `UUID` of the table.

Usually, the `UUID` is generated automatically, but the user can also explicitly specify the `UUID` in the same way when creating the table (this is not recommended). To display the `SHOW CREATE` query with the `UUID` you can use setting `show_table_uuid_in_table_create_query_if_not_nil`. For example:

```
CREATE TABLE name UUID '28f1c61c-2970-457a-bffe-454156ddcef' (n UInt64) ENGINE = ...;
```

### RENAME TABLE

`RENAME` queries are performed without changing `UUID` and moving table data. These queries do not wait for the completion of queries using the table and are executed instantly.

### DROP/DETACH TABLE

On `DROP TABLE` no data is removed, database Atomic just marks table as dropped by moving metadata to `/clickhouse_path/metadata_dropped/` and notifies background thread. Delay before final table data deletion is specified by the `database_atomic_delay_before_drop_table_sec` setting.

You can specify synchronous mode using `SYNC` modifier. Use the `database_atomic_wait_for_drop_and_detach_synchronously` setting to do this. In this case `DROP` waits for running `SELECT`, `INSERT` and other queries which are using the table to finish. Table will be actually removed when it's not in use.

## EXCHANGE TABLES/DICTIONARIES

`EXCHANGE` query swaps tables or dictionaries atomically. For instance, instead of this non-atomic operation:

```
RENAME TABLE new_table TO tmp, old_table TO new_table, tmp TO old_table;
```

you can use one atomic query:

```
EXCHANGE TABLES new_table AND old_table;
```

## ReplicatedMergeTree in Atomic Database

For `ReplicatedMergeTree` tables, it is recommended not to specify engine parameters - path in ZooKeeper and replica name. In this case, configuration parameters `default_replica_path` and `default_replica_name` will be used. If you want to specify engine parameters explicitly, it is recommended to use `{uuid}` macros. This is useful so that unique paths are automatically generated for each table in ZooKeeper.

## See Also

- `system.databases` system table

## SQLite

Allows to connect to `SQLite` database and perform `INSERT` and `SELECT` queries to exchange data between ClickHouse and SQLite.

## Creating a Database

```
CREATE DATABASE sqlite_database
ENGINE = SQLite('db_path')
```

### Engine Parameters

- `db_path` — Path to a file with SQLite database.

## Data Types Support

SQLite	ClickHouse
INTEGER	<code>Int32</code>
REAL	<code>Float32</code>
TEXT	<code>String</code>

**SQLite****ClickHouse**

BLOB

String

## Specifics and Recommendations

SQLite stores the entire database (definitions, tables, indices, and the data itself) as a single cross-platform file on a host machine. During writing SQLite locks the entire database file, therefore write operations are performed sequentially. Read operations can be multitasked.  
SQLite does not require service management (such as startup scripts) or access control based on `GRANT` and passwords. Access control is handled by means of file-system permissions given to the database file itself.

## Usage Example

Database in ClickHouse, connected to the SQLite:

```
CREATE DATABASE sqlite_db ENGINE = SQLite('sqlite.db');
SHOW TABLES FROM sqlite_db;
```

name
table1
table2

Shows the tables:

```
SELECT * FROM sqlite_db.table1;
```

col1	col2
line1	1
line2	2
line3	3

Inserting data into SQLite table from ClickHouse table:

```
CREATE TABLE clickhouse_table(`col1` String, `col2` Int16) ENGINE = MergeTree() ORDER BY col2;
INSERT INTO clickhouse_table VALUES ('text',10);
INSERT INTO sqlite_db.table1 SELECT * FROM clickhouse_table;
SELECT * FROM sqlite_db.table1;
```

col1	col2
line1	1
line2	2
line3	3
text	10

## PostgreSQL

Allows to connect to databases on a remote PostgreSQL server. Supports read and write operations (`SELECT` and `INSERT` queries) to exchange data between ClickHouse and PostgreSQL.

Gives the real-time access to table list and table structure from remote PostgreSQL with the help of `SHOW TABLES` and `DESCRIBE TABLE` queries.

Supports table structure modifications (`ALTER TABLE ... ADD|DROP COLUMN`). If `use_table_cache` parameter (see the Engine Parameters below) is set to `1`, the table structure is cached and not checked for being modified, but can be updated with `DETACH` and `ATTACH` queries.

## Creating a Database

```
CREATE DATABASE test_database  
ENGINE = PostgreSQL('host:port', 'database', 'user', 'password'[, `schema`, `use_table_cache`]);
```

### Engine Parameters

- `host:port` — PostgreSQL server address.
- `database` — Remote database name.
- `user` — PostgreSQL user.
- `password` — User password.
- `schema` — PostgreSQL schema.
- `use_table_cache` — Defines if the database table structure is cached or not. Optional. Default value: `0`.

## Data Types Support

PostgreSQL	ClickHouse
DATE	Date
TIMESTAMP	DateTime
REAL	Float32
DOUBLE	Float64
DECIMAL, NUMERIC	Decimal
SMALLINT	Int16
INTEGER	Int32
BIGINT	Int64
SERIAL	UInt32
BIGSERIAL	UInt64
TEXT, CHAR	String
INTEGER	Nullable(Int32)
ARRAY	Array

# Examples of Use

Database in ClickHouse, exchanging data with the PostgreSQL server:

```
CREATE DATABASE test_database  
ENGINE = PostgreSQL('postgres1:5432', 'test_database', 'postgres', 'mysecretpassword', 1);
```

```
SHOW DATABASES;
```

name
default
test_database
system

```
SHOW TABLES FROM test_database;
```

name
test_table

Reading data from the PostgreSQL table:

```
SELECT * FROM test_database.test_table;
```

id	value
1	1
2	2

Writing data to the PostgreSQL table:

```
INSERT INTO test_database.test_table VALUES (3,4);  
SELECT * FROM test_database.test_table;
```

int_id	value
1	1
3	4

Consider the table structure was modified in PostgreSQL:

```
postgre> ALTER TABLE test_table ADD COLUMN data Text
```

As the `use_table_cache` parameter was set to `1` when the database was created, the table structure in ClickHouse was cached and therefore not modified:

```
DESCRIBE TABLE test_database.test_table;
```

name	type
id	Nullable(Integer)
value	Nullable(Integer)

After detaching the table and attaching it again, the structure was updated:

```
DETACH TABLE test_database.test_table;
ATTACH TABLE test_database.test_table;
DESCRIBE TABLE test_database.test_table;
```

name	type
id	Nullable(Integer)
value	Nullable(Integer)
data	Nullable(String)

## [experimental] Replicated

The engine is based on the [Atomic](#) engine. It supports replication of metadata via DDL log being written to ZooKeeper and executed on all of the replicas for a given database.

One ClickHouse server can have multiple replicated databases running and updating at the same time. But there can't be multiple replicas of the same replicated database.

### Creating a Database

```
CREATE DATABASE testdb ENGINE = Replicated('zoo_path', 'shard_name', 'replica_name') [SETTINGS ...]
```

#### Engine Parameters

- `zoo_path` — ZooKeeper path. The same ZooKeeper path corresponds to the same database.
- `shard_name` — Shard name. Database replicas are grouped into shards by `shard_name`.
- `replica_name` — Replica name. Replica names must be different for all replicas of the same shard.

### Warning

For [ReplicatedMergeTree](#) tables if no arguments provided, then default arguments are used:

`/clickhouse/tables/{uuid}/{shard}` and `{replica}`. These can be changed in the server settings

`default_replica_path` and `default_replica_name`. Macro `{uuid}` is unfolded to table's `uuid`, `{shard}` and `{replica}` are unfolded to values from server config, not from database engine arguments. But in the future, it will be possible to use `shard_name` and `replica_name` of Replicated database.

## Specifics and Recommendations

DDL queries with `Replicated` database work in a similar way to [ON CLUSTER](#) queries, but with minor differences.

First, the DDL request tries to execute on the initiator (the host that originally received the request from the user). If the request is not fulfilled, then the user immediately receives an error, other hosts do not try to fulfill it. If the request has been successfully completed on the initiator, then all other hosts will

automatically retry until they complete it. The initiator will try to wait for the query to be completed on other hosts (no longer than `distributed_ddl_task_timeout`) and will return a table with the query execution statuses on each host.

The behavior in case of errors is regulated by the `distributed_ddl_output_mode` setting, for a `Replicated` database it is better to set it to `null_status_on_timeout` — i.e. if some hosts did not have time to execute the request for `distributed_ddl_task_timeout`, then do not throw an exception, but show the `NULL` status for them in the table.

The `system.clusters` system table contains a cluster named like the replicated database, which consists of all replicas of the database. This cluster is updated automatically when creating/deleting replicas, and it can be used for `Distributed` tables.

When creating a new replica of the database, this replica creates tables by itself. If the replica has been unavailable for a long time and has lagged behind the replication log — it checks its local metadata with the current metadata in ZooKeeper, moves the extra tables with data to a separate non-replicated database (so as not to accidentally delete anything superfluous), creates the missing tables, updates the table names if they have been renamed. The data is replicated at the `ReplicatedMergeTree` level, i.e. if the table is not replicated, the data will not be replicated (the database is responsible only for metadata).

`ALTER TABLE ATTACH|FETCH|DROP|DROP DETACHED|DETACH PARTITION|PART` queries are allowed but not replicated. The database engine will only add/fetch/remove the partition/part to the current replica. However, if the table itself uses a `Replicated` table engine, then the data will be replicated after using `ATTACH`.

## Usage Example

Creating a cluster with three hosts:

```
node1 :) CREATE DATABASE r ENGINE=Replicated('some/path/r','shard1','replica1');
node2 :) CREATE DATABASE r ENGINE=Replicated('some/path/r','shard1','other_replica');
node3 :) CREATE DATABASE r ENGINE=Replicated('some/path/r','other_shard','{replica}');
```

Running the DDL-query:

```
CREATE TABLE r.rmt (n UInt64) ENGINE=ReplicatedMergeTree ORDER BY n;
```

hosts	status	error	num_hosts_remaining	num_hosts_active
shard1 replica1	0	2	0	
shard1 other_replica	0	1	0	
other_shard r1	0	0	0	

Showing the system table:

```
SELECT cluster, shard_num, replica_num, host_name, host_address, port, is_local
FROM system.clusters WHERE cluster='r';
```

cluster	shard_num	replica_num	host_name	host_address	port	is_local
r	1	1	node3	127.0.0.1	9002	0
r	2	1	node2	127.0.0.1	9001	0
r	2	2	node1	127.0.0.1	9000	1

Creating a distributed table and inserting the data:

```
node2 :) CREATE TABLE r.d (n UInt64) ENGINE=Distributed('r','r','rmt', n % 2);
node3 :) INSERT INTO r SELECT * FROM numbers(10);
node1 :) SELECT materialize(hostName()) AS host, groupArray(n) FROM r.d GROUP BY host;
```

hosts	groupArray(n)
node1	[1,3,5,7,9]
node2	[0,2,4,6,8]

Adding replica on the one more host:

```
node4 :) CREATE DATABASE r ENGINE=Replicated('some/path/r','other_shard','r2');
```

The cluster configuration will look like this:

cluster	shard_num	replica_num	host_name	host_address	port	is_local
r	1	1	node3	127.0.0.1	9002	0
r	1	2	node4	127.0.0.1	9003	0
r	2	1	node2	127.0.0.1	9001	0
r	2	2	node1	127.0.0.1	9000	1

The distributed table also will get data from the new host:

```
node2 :) SELECT materialize(hostName()) AS host, groupArray(n) FROM r.d GROUP BY host;
```

hosts	groupArray(n)
node2	[1,3,5,7,9]
node4	[0,2,4,6,8]

## SQL Reference

ClickHouse supports the following types of queries:

- [SELECT](#)
- [INSERT INTO](#)
- [CREATE](#)
- [ALTER](#)
- [Other types of queries](#)

## ClickHouse SQL Statements

Statements represent various kinds of action you can perform using SQL queries. Each kind of statement has its own syntax and usage details that are described separately:

- [SELECT](#)
- [INSERT INTO](#)
- [CREATE](#)

- **ALTER**
  - **SYSTEM**
  - **SHOW**
  - **GRANT**
  - **REVOKE**
  - **ATTACH**
  - **CHECK TABLE**
  - **DESCRIBE TABLE**
  - **DETACH**
  - **DROP**
  - **EXISTS**
  - **KILL**
  - **OPTIMIZE**
  - **RENAME**
  - **SET**
  - **SET ROLE**
  - **TRUNCATE**
  - **USE**
  - **EXPLAIN**
- 

## SELECT Query

`SELECT` queries perform data retrieval. By default, the requested data is returned to the client, while in conjunction with `INSERT INTO` it can be forwarded to a different table.

### Syntax

```
[WITH expr_list|(subquery)]
SELECT [DISTINCT [ON (column1, column2, ...)]] expr_list
[FROM [db.]table | (subquery) | table_function] [FINAL]
[SAMPLE sample_coeff]
[ARRAY JOIN ...]
[GLOBAL] [ANY|ALL|ASOF] [INNER|LEFT|RIGHT|FULL|CROSS] [OUTER|SEMI|ANTI] JOIN (subquery)|table (ON
<expr_list>)|(USING <column_list>)
[PREWHERE expr]
[WHERE expr]
[GROUP BY expr_list] [WITH ROLLUP|WITH CUBE] [WITH TOTALS]
[HAVING expr]
[ORDER BY expr_list] [WITH FILL] [FROM expr] [TO expr] [STEP expr]
[LIMIT [offset_value, ]n BY columns]
[LIMIT [n, ]m] [WITH TIES]
[SETTINGS ...]
[UNION ...]
[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](#)
- [SELECT clause](#)
- [DISTINCT clause](#)
- [FROM clause](#)
- [SAMPLE clause](#)
- [JOIN clause](#)
- [PREWHERE clause](#)
- [WHERE clause](#)
- [GROUP BY clause](#)
- [LIMIT BY clause](#)
- [HAVING clause](#)
- [LIMIT clause](#)
- [OFFSET clause](#)
- [UNION clause](#)
- [INTERSECT clause](#)
- [EXCEPT 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 ...`

## COLUMNS expression

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
```

aa	ab
1	1

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
```

aa	ab	bc	toTypeName(bc)
1	1	1	Int8

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 does not 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 does not 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` does not 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 do not 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`, `ORDER BY`, and `LIMIT BY` clauses can support positional arguments. To enable this, switch on the `enable_positional_arguments` setting. Then, for example, `ORDER BY 1,2` will be sorting rows in the table on the first and then the second column.

## 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.

# SELECT modifiers

You can use the following modifiers in `SELECT` queries.

## APPLY

Allows you to invoke some function for each row returned by an outer table expression of a query.

### Syntax:

```
SELECT <expr> APPLY( <func> ) FROM [db.]table_name
```

### Example:

```
CREATE TABLE columns_transformers (i Int64, j Int16, k Int64) ENGINE = MergeTree ORDER by (i);
INSERT INTO columns_transformers VALUES (100, 10, 324), (120, 8, 23);
SELECT * APPLY(sum) FROM columns_transformers;
```

sum(i)	sum(j)	sum(k)
220	18	347

## EXCEPT

Specifies the names of one or more columns to exclude from the result. All matching column names are omitted from the output.

### Syntax:

```
SELECT <expr> EXCEPT ( col_name1 [, col_name2, col_name3, ...] ) FROM [db.]table_name
```

### Example:

```
SELECT * EXCEPT (i) from columns_transformers;
```

j	k
10	324
8	23

## REPLACE

Specifies one or more [expression aliases](#). Each alias must match a column name from the `SELECT *` statement. In the output column list, the column that matches the alias is replaced by the expression in that `REPLACE`.

This modifier does not change the names or order of columns. However, it can change the value and the value type.

### Syntax:

```
SELECT <expr> REPLACE( <expr> AS col_name ) from [db.]table_name
```

### Example:

```
SELECT * REPLACE(i + 1 AS i) from columns_transformers;
```

i	j	k
101	10	324
121	8	23

## Modifier Combinations

You can use each modifier separately or combine them.

### Examples:

Using the same modifier multiple times.

```
SELECT COLUMNS('[jk]') APPLY(toString) APPLY(length) APPLY(max) from columns_transformers;
```

max(length(toString(j)))	max(length(toString(k)))
2	3

Using multiple modifiers in a single query.

```
SELECT * REPLACE(i + 1 AS i) EXCEPT (j) APPLY(sum) from columns_transformers;
```

sum(plus(i, 1))	sum(k)
222	347

## SETTINGS in SELECT Query

You can specify the necessary settings right in the `SELECT` query. The setting value is applied only to this query and is reset to default or previous value after the query is executed.

Other ways to make settings see [here](#).

### Example

```
SELECT * FROM some_table SETTINGS optimize_read_in_order=1, cast_keep_nullable=1;
```

## ALL Clause

If there are multiple matching rows in the table, then `ALL` returns all of them. `SELECT ALL` is identical to `SELECT` without `DISTINCT`. If both `ALL` and `DISTINCT` specified, exception will be thrown.

`ALL` can also be specified inside aggregate function with the same effect(noop), for instance:

```
SELECT sum(ALL number) FROM numbers(10);
```

equals to

```
SELECT sum(number) FROM numbers(10);
```

## 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]), ('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]	
World	3	30	[3,4,5]	[30,40,50]	1
World	4	40	[3,4,5]	[30,40,50]	
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.

You can specify the list of columns that must have unique values: `SELECT DISTINCT ON (column1, column2,...)`. If the columns are not specified, all of them are taken into consideration.

Consider the table:

a	b	c
1	1	1
1	1	1
2	2	2
2	2	2
1	1	2
1	2	2

Using `DISTINCT` without specifying columns:

```
SELECT DISTINCT * FROM t1;
```

a	b	c
1	1	1
2	2	2
1	1	2
1	2	2

Using `DISTINCT` with specified columns:

```
SELECT DISTINCT ON (a,b) * FROM t1;
```

a	b	c
1	1	1
2	2	2
1	2	2

## DISTINCT and ORDER BY

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.

Consider the table:

a	b
2	1
1	2
3	3
2	4

Selecting data:

```
SELECT DISTINCT a FROM t1 ORDER BY b ASC;
```

a
2
1
3

Selecting data with the different sorting direction:

```
SELECT DISTINCT a FROM t1 ORDER BY b DESC;
```

a
3
1
2

Row 2, 4 was cut before sorting.

Take this implementation specificity into account when programming queries.

## 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.

## EXCEPT Clause

The `EXCEPT` clause returns only those rows that result from the first query without the second. The queries must match the number of columns, order, and type. The result of `EXCEPT` can contain duplicate rows.

Multiple `EXCEPT` statements are executed left to right if parenthesis are not specified. The `EXCEPT` operator has the same priority as the `UNION` clause and lower priority than the `INTERSECT` clause.

```
SELECT column1 [, column2 ]
FROM table1
[WHERE condition]
```

`EXCEPT`

```
SELECT column1 [, column2 ]
FROM table2
[WHERE condition]
```

The condition could be any expression based on your requirements.

## Examples

Query:

```
SELECT number FROM numbers(1,10) EXCEPT SELECT number FROM numbers(3,6);
```

Result:

number
1
2
9
10

Query:

```
CREATE TABLE t1(one String, two String, three String) ENGINE=Memory();
CREATE TABLE t2(four String, five String, six String) ENGINE=Memory();

INSERT INTO t1 VALUES ('q', 'm', 'b'), ('s', 'd', 'f'), ('l', 'p', 'o'), ('s', 'd', 'f'), ('s', 'd', 'f'), ('k', 't', 'd'), ('l', 'p', 'o');
INSERT INTO t2 VALUES ('q', 'm', 'b'), ('b', 'd', 'k'), ('s', 'y', 't'), ('s', 'd', 'f'), ('m', 'f', 'o'), ('k', 'k', 'd');

SELECT * FROM t1 EXCEPT SELECT * FROM t2;
```

Result:

one	two	three
l	p	o
k	t	d
l	p	o

## See Also

- [UNION](#)
- [INTERSECT](#)

## 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` [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](#). 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.

Now `SELECT` queries with `FINAL` are executed in parallel and slightly faster. But there are drawbacks (see below). The [max\\_final\\_threads](#) setting limits the number of threads used.

## Drawbacks

Queries that use `FINAL` are executed slightly slower than similar queries that do not, because:

- 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 have'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 expression”.
- 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.

When you want to group data in the table by column numbers instead of column names, enable the setting `enable_positional_arguments`.

### 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 ROLLUP Modifier

`WITH ROLLUP` modifier is used to calculate subtotals for the key expressions, based on their order in the `GROUP BY` list. The subtotals rows are added after the result table.

The subtotals are calculated in the reverse order: at first subtotals are calculated for the last key expression in the list, then for the previous one, and so on up to the first key expression.

In the subtotals rows the values of already "grouped" key expressions are set to 0 or empty line.

### Note

Mind that **HAVING** clause can affect the subtotals results.

### Example

Consider the table `t`:

year	month	day
2019	1	5
2019	1	15
2020	1	5
2020	1	15
2020	10	5
2020	10	15

Query:

```
SELECT year, month, day, count(*) FROM t GROUP BY year, month, day WITH ROLLUP;
```

As `GROUP BY` section has three key expressions, the result contains four tables with subtotals "rolled up" from right to left:

- `GROUP BY year, month, day`;
- `GROUP BY year, month` (and `day` column is filled with zeros);

- `GROUP BY` year (now month, day columns are both filled with zeros);
- and totals (and all three key expression columns are zeros).

year	month	day	count()
2020	10	15	1
2020	1	5	1
2019	1	5	1
2020	1	15	1
2019	1	15	1
2020	10	5	1

year	month	day	count()
2019	1	0	2
2020	1	0	2
2020	10	0	2

year	month	day	count()
2019	0	0	2
2020	0	0	4

year	month	day	count()
0	0	0	6

## WITH CUBE Modifier

`WITH CUBE` modifier is used to calculate subtotals for every combination of the key expressions in the `GROUP BY` list. The subtotals rows are added after the result table.

In the subtotals rows the values of all "grouped" key expressions are set to 0 or empty line.

### Note

Mind that **HAVING** clause can affect the subtotals results.

### Example

Consider the table t:

year	month	day
2019	1	5
2019	1	15
2020	1	5
2020	1	15
2020	10	5
2020	10	15

Query:

```
SELECT year, month, day, count(*) FROM t GROUP BY year, month, day WITH CUBE;
```

As `GROUP BY` section has three key expressions, the result contains eight tables with subtotals for all key expression combinations:

- `GROUP BY` year, month, day
- `GROUP BY` year, month
- `GROUP BY` year, day

- GROUP BY year
- GROUP BY month, day
- GROUP BY month
- GROUP BY day
- and totals.

Columns, excluded from GROUP BY, are filled with zeros.

year	month	day	count()
2020	10	15	1
2020	1	5	1
2019	1	5	1
2020	1	15	1
2019	1	15	1
2020	10	5	1

year	month	day	count()
2019	1	0	2
2020	1	0	2
2020	10	0	2

year	month	day	count()
2020	0	5	2
2019	0	5	1
2020	0	15	2
2019	0	15	1

year	month	day	count()
2019	0	0	2
2020	0	0	4

year	month	day	count()
0	1	5	2
0	10	15	1
0	10	5	1
0	1	15	2

year	month	day	count()
0	1	0	4
0	10	0	2

year	month	day	count()
0	0	5	3
0	0	15	3

year	month	day	count()
0	0	0	6

## 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 do not 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
```

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.

## 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 Optimization Depending on Table Sorting Key

The aggregation can be performed more effectively, if a table is sorted by some key, and `GROUP BY` expression contains at least prefix of sorting key or injective functions. In this case when a new key is read from table, the in-between result of aggregation can be finalized and sent to client. This behaviour is switched on by the `optimize_aggregation_in_order` setting. Such optimization reduces memory usage during aggregation, but in some cases may slow down the query execution.

## 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` does not have `LIMIT`, do not 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.

# INTERSECT Clause

The `INTERSECT` clause returns only those rows that result from both the first and the second queries. The queries must match the number of columns, order, and type. The result of `INTERSECT` can contain duplicate rows.

Multiple `INTERSECT` statements are executes left to right if parenthesis are not specified. The `INTERSECT` operator has a higher priority than the `UNION` and `EXCEPT` clause.

```
SELECT column1 [, column2 ]
FROM table1
[WHERE condition]
```

`INTERSECT`

```
SELECT column1 [, column2 ]
FROM table2
[WHERE condition]
```

The condition could be any expression based on your requirements.

## Examples

Query:

```
SELECT number FROM numbers(1,10) INTERSECT SELECT number FROM numbers(3,6);
```

Result:

number
3
4
5
6
7
8

Query:

```
CREATE TABLE t1(one String, two String, three String) ENGINE=Memory();
CREATE TABLE t2(four String, five String, six String) ENGINE=Memory();

INSERT INTO t1 VALUES ('q', 'm', 'b'), ('s', 'd', 'f'), ('l', 'p', 'o'), ('s', 'd', 'f'), ('s', 'd', 'f'), ('k', 't', 'd'), ('l', 'p', 'o');
INSERT INTO t2 VALUES ('q', 'm', 'b'), ('b', 'd', 'k'), ('s', 'y', 't'), ('s', 'd', 'f'), ('m', 'f', 'o'), ('k', 'k', 'd');

SELECT * FROM t1 INTERSECT SELECT * FROM t2;
```

Result:

one	two	three
q	m	b
s	d	f
s	d	f
s	d	f

## See Also

- [UNION](#)

- EXCEPT

## 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.

## 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] [INNER|LEFT|RIGHT|FULL|CROSS] [OUTER|SEMI|ANTI|ANY|ASOF] 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.
- `LEFT ANY JOIN`, `RIGHT ANY JOIN` and `INNER ANY JOIN`, partially (for opposite side of `LEFT` and `RIGHT`) or completely (for `INNER` and `FULL`) disables the cartesian product for standard `JOIN` types.

- ASOF JOIN and LEFT ASOF JOIN, joining sequences with a non-exact match. ASOF JOIN usage is described below.

## Note

When **join\_algorithm** is set to `partial_merge`, RIGHT JOIN and FULL JOIN are supported only with ALL strictness (`SEMI`, `ANTI`, `ANY`, and `ASOF` are not supported).

## Settings

The default join type can be overridden using `join_default_strictness` setting.

The behavior of ClickHouse server for ANY JOIN operations depends on the `any_join_distinct_right_table_keys` setting.

### See also

- [join\\_algorithm](#)
- [join\\_any\\_take\\_last\\_row](#)
- [join\\_use\\_nulls](#)
- [partial\\_merge\\_join\\_optimizations](#)
- [partial\\_merge\\_join\\_rows\\_in\\_right\\_blocks](#)
- [join\\_on\\_disk\\_max\\_files\\_to\\_merge](#)
- [any\\_join\\_distinct\\_right\\_table\\_keys](#)

## ON Section Conditions

An ON section can contain several conditions combined using the AND operator. Conditions specifying join keys must refer both left and right tables and must use the equality operator. Other conditions may use other logical operators but they must refer either the left or the right table of a query.

Rows are joined if the whole complex condition is met. If the conditions are not met, still rows may be included in the result depending on the JOIN type. Note that if the same conditions are placed in a WHERE section and they are not met, then rows are always filtered out from the result.

## Note

The OR operator inside an ON section is not supported yet.

## Note

If a condition refers columns from different tables, then only the equality operator (=) is supported so far.

### Example

Consider table\_1 and table\_2:

Id	name		Id	text	scores
1	A		1	Text A	10
2	B		1	Another text A	12
3	C		2	Text B	15

Query with one join key condition and an additional condition for `table_2`:

```
SELECT name, text FROM table_1 LEFT OUTER JOIN table_2
  ON table_1.Id = table_2.Id AND startsWith(table_2.text, 'Text');
```

Note that the result contains the row with the name C and the empty text column. It is included into the result because an `OUTER` type of a join is used.

name	text
A	Text A
B	Text B
C	

Query with `INNER` type of a join and multiple conditions:

```
SELECT name, text, scores FROM table_1 INNER JOIN table_2
  ON table_1.Id = table_2.Id AND table_2.scores > 10 AND startsWith(table_2.text, 'Text');
```

Result:

name	text	scores
B	Text B	15

## ASOF JOIN Usage

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

Algorithm requires the special column in tables. This column:

- Must contain an ordered sequence.
- Can be one of the following types: `Int`, `UInt`, `Float`, `Date`, `DateTime`, `Decimal`.
- Can't be the only column in the `JOIN` clause.

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 is 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_2_2	12:30	42
event_1_2	13:00	42	event_2_3	13:00	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.

## Implicit Type Conversion

`INNER JOIN`, `LEFT JOIN`, `RIGHT JOIN`, and `FULL JOIN` queries support the implicit type conversion for "join keys". However the query can not be executed, if join keys from the left and the right tables cannot be converted to a single type (for example, there is no data type that can hold all values from both `UInt64` and `Int64`, or `String` and `Int32`).

### Example

Consider the table `t_1`:

a	b	toTypeName(a)	toTypeName(b)
1	1	UInt16	UInt8
2	2	UInt16	UInt8

and the table `t_2`:

a	b	toTypeName(a)	toTypeName(b)
-1	1	Int16	Nullable(Int64)
1	-1	Int16	Nullable(Int64)
1	1	Int16	Nullable(Int64)

The query

```
SELECT a, b, toTypeName(a), toTypeName(b) FROM t_1 FULL JOIN t_2 USING (a, b);
```

returns the set:

a	b	toTypeName(a)	toTypeName(b)
1	1	Int32	Nullable(Int64)
2	2	Int32	Nullable(Int64)
-1	1	Int32	Nullable(Int64)
1	-1	Int32	Nullable(Int64)

## 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 `NUL`L 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. If `join_algorithm = 'auto'` is enabled, then after some threshold of memory consumption, ClickHouse falls back to [merge](#) join algorithm. For JOIN algorithms description see the [join\\_algorithm](#) setting.

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.

### Note

The number of rows in the result set can also depend on the `limit` setting.

## LIMIT ... WITH TIES Modifier

When you set `WITH TIES` modifier for `LIMIT n[,m]` and specify `ORDER BY expr_list`, you will get in result first `n` or `n,m` rows and all rows with same `ORDER BY` fields values equal to row at position `n` for `LIMIT n` and `m` for `LIMIT n,m`.

This modifier also can be combined with `ORDER BY ... WITH FILL modifier`.

For example, the following query

```
SELECT * FROM (
  SELECT number%50 AS n FROM numbers(100)
) ORDER BY n LIMIT 0,5
```

returns

n
0
0
1
1
2

but after apply `WITH TIES` modifier

```
SELECT * FROM (
    SELECT number%50 AS n FROM numbers(100)
) ORDER BY n LIMIT 0,5 WITH TIES
```

it returns another rows set

n
0
0
1
1
2
2
2

cause row number 6 have same value "2" for field n as row number 5

## 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.

If you want to use column numbers instead of column names in the `LIMIT BY` clause, enable the setting `enable_positional_arguments`.

## 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 1, 2 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
```

## OFFSET FETCH Clause

`OFFSET` and `FETCH` allow you to retrieve data by portions. They specify a row block which you want to get by a single query.

```
OFFSET offset_row_count {ROW | ROWS} [FETCH {FIRST | NEXT} fetch_row_count {ROW | ROWS} {ONLY | WITH TIES}]
```

The `offset_row_count` or `fetch_row_count` value can be a number or a literal constant. You can omit `fetch_row_count`; by default, it equals to 1.

`OFFSET` specifies the number of rows to skip before starting to return rows from the query result set.

The `FETCH` specifies the maximum number of rows that can be in the result of a query.

The `ONLY` option is used to return rows that immediately follow the rows omitted by the `OFFSET`. In this case the `FETCH` is an alternative to the `LIMIT` clause. For example, the following query

```
SELECT * FROM test_fetch ORDER BY a OFFSET 1 ROW FETCH FIRST 3 ROWS ONLY;
```

is identical to the query

```
SELECT * FROM test_fetch ORDER BY a LIMIT 3 OFFSET 1;
```

The **WITH TIES** option is used to return any additional rows that tie for the last place in the result set according to the **ORDER BY** clause. For example, if `fetch_row_count` is set to 5 but two additional rows match the values of the **ORDER BY** columns in the fifth row, the result set will contain seven rows.

## Note

According to the standard, the **OFFSET** clause must come before the **FETCH** clause if both are present.

## Note

The real offset can also depend on the **offset** setting.

## Examples

Input table:

a	b
1	1
2	1
3	4
1	3
5	4
0	6
5	7

Usage of the **ONLY** option:

```
SELECT * FROM test_fetch ORDER BY a OFFSET 3 ROW FETCH FIRST 3 ROWS ONLY;
```

Result:

a	b
2	1
3	4
5	4

Usage of the **WITH TIES** option:

```
SELECT * FROM test_fetch ORDER BY a OFFSET 3 ROW FETCH FIRST 3 ROWS WITH TIES;
```

Result:

a	b
2	1
3	4
5	4
5	7

## 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`.

If you want to sort by column numbers instead of column names, enable the setting [enable\\_positional\\_arguments](#).

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.

Collate is supported in **LowCardinality**, **Nullable**, **Array** and **Tuple**.

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.

## Collation Examples

Example only with **String** values:

Input table:

X	S
1	bca
2	ABC
3	123a
4	abc
5	BCA

Query:

```
SELECT * FROM collate_test ORDER BY s ASC COLLATE 'en';
```

Result:

X	S
3	123a
4	abc
2	ABC
1	bca
5	BCA

Example with **Nullable**:

Input table:

X	S
1	bca
2	NULL
3	ABC
4	123a
5	abc
6	NULL
7	BCA

Query:

```
SELECT * FROM collate_test ORDER BY s ASC COLLATE 'en';
```

Result:

X	S
4	123a
5	abc
3	ABC
1	bca
7	BCA
6	NULL
2	NULL

Example with **Array**:

Input table:

X	S
1	['Z']
2	['z']
3	['a']
4	['A']
5	['z','a']
6	['z','a','a']
7	[]

Query:

```
SELECT * FROM collate_test ORDER BY s ASC COLLATE 'en';
```

Result:

X	S
7	[]
3	['a']
4	['A']
2	['z']
5	['z','a']
6	['z','a','a']
1	['Z']

Example with **LowCardinality** string:

Input table:

X	S
1	Z
2	z
3	a
4	A
5	za
6	zaa
7	

Query:

```
SELECT * FROM collate_test ORDER BY s ASC COLLATE 'en';
```

Result:

X	S
7	a
3	A
4	z
2	Z
1	za
5	zaa
6	zaa

Example with **Tuple**:

X	S
1	(1,'Z')
2	(1,'z')
3	(1,'a')
4	(2,'z')
5	(1,'A')
6	(2,'Z')
7	(2,'A')

Query:

```
SELECT * FROM collate_test ORDER BY s ASC COLLATE 'en';
```

Result:

X	S
3	(1,'a')
5	(1,'A')
2	(1,'z')
1	(1,'Z')
7	(2,'A')
4	(2,'z')
6	(2,'Z')

## 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.

# Optimization of Data Reading

If `ORDER BY` expression has a prefix that coincides with the table sorting key, you can optimize the query by using the `optimize_read_in_order` setting.

When the `optimize_read_in_order` setting is enabled, the ClickHouse server uses the table index and reads the data in order of the `ORDER BY` key. This allows to avoid reading all data in case of specified `LIMIT`. So queries on big data with small limit are processed faster.

Optimization works with both `ASC` and `DESC` and does not work together with `GROUP BY` clause and `FINAL` modifier.

When the `optimize_read_in_order` setting is disabled, the ClickHouse server does not use the table index while processing `SELECT` queries.

Consider disabling `optimize_read_in_order` manually, when running queries that have `ORDER BY` clause, large `LIMIT` and `WHERE` condition that requires to read huge amount of records before queried data is found.

Optimization is supported in the following table engines:

- `MergeTree`
- `Merge`, `Buffer`, and `MaterializedView` table engines over `MergeTree`-engine tables

In `MaterializedView`-engine tables the optimization works with views like `SELECT ... FROM merge_tree_table ORDER BY pk`. But it is not supported in the queries like `SELECT ... FROM view ORDER BY pk` if the view query does not have the `ORDER BY` clause.

## ORDER BY Expr WITH FILL Modifier

This modifier also can be combined with `LIMIT ... WITH TIES` modifier.

`WITH FILL` modifier can be set after `ORDER BY expr` with optional `FROM expr`, `TO expr` and `STEP expr` parameters. All missed values of `expr` column will be filled sequentially and other columns will be filled as defaults.

To fill multiple columns, add `WITH FILL` modifier with optional parameters after each field name in `ORDER BY` section.

```
ORDER BY expr [WITH FILL] [FROM const_expr] [TO const_expr] [STEP const_numeric_expr], ... exprN [WITH FILL]
[FROM expr] [TO expr] [STEP numeric_expr]
```

`WITH FILL` can be applied for fields with Numeric (all kinds of float, decimal, int) or Date/DateTime types. When applied for String fields, missed values are filled with empty strings.

When `FROM const_expr` not defined sequence of filling use minimal `expr` field value from `ORDER BY`.

When `TO const_expr` not defined sequence of filling use maximum `expr` field value from `ORDER BY`.

When `STEP const_numeric_expr` defined then `const_numeric_expr` interprets as is for numeric types as days for Date type and as seconds for DateTime type.

When `STEP const_numeric_expr` omitted then sequence of filling use 1.0 for numeric type, 1 day for Date type and 1 second for DateTime type.

Example of a query without `WITH FILL`:

```
SELECT n, source FROM (
    SELECT toFloat32(number % 10) AS n, 'original' AS source
    FROM numbers(10) WHERE number % 3 = 1
) ORDER BY n;
```

Result:

n	source
1	original
4	original
7	original

Same query after applying WITH FILL modifier:

```
SELECT n, source FROM (
  SELECT toFloat32(number % 10) AS n, 'original' AS source
  FROM numbers(10) WHERE number % 3 = 1
) ORDER BY n WITH FILL FROM 0 TO 5.51 STEP 0.5;
```

Result:

n	source
0	
0.5	
1	original
1.5	
2	
2.5	
3	
3.5	
4	original
4.5	
5	
5.5	
7	original

For the case with multiple fields ORDER BY field2 WITH FILL, field1 WITH FILL order of filling will follow the order of fields in the ORDER BY clause.

Example:

```
SELECT
  toDate((number * 10) * 86400) AS d1,
  toDate(number * 86400) AS d2,
  'original' AS source
FROM numbers(10)
WHERE (number % 3) = 1
ORDER BY
  d2 WITH FILL,
  d1 WITH FILL STEP 5;
```

Result:

d1	d2	source
1970-01-11	1970-01-02	original
1970-01-01	1970-01-03	
1970-01-01	1970-01-04	
1970-02-10	1970-01-05	original
1970-01-01	1970-01-06	
1970-01-01	1970-01-07	
1970-03-12	1970-01-08	original

Field d1 does not fill in and use the default value cause we do not have repeated values for d2 value, and the sequence for d1 can't be properly calculated.

The following query with the changed field in ORDER BY:

```

SELECT
    toDate((number * 10) * 86400) AS d1,
    toDate(number * 86400) AS d2,
    'original' AS source
FROM numbers(10)
WHERE (number % 3) = 1
ORDER BY
    d1 WITH FILL STEP 5,
    d2 WITH FILL;

```

Result:

d1	d2	source
1970-01-11	1970-01-02	original
1970-01-16	1970-01-01	
1970-01-21	1970-01-01	
1970-01-26	1970-01-01	
1970-01-31	1970-01-01	
1970-02-05	1970-01-01	
1970-02-10	1970-01-05	original
1970-02-15	1970-01-01	
1970-02-20	1970-01-01	
1970-02-25	1970-01-01	
1970-03-02	1970-01-01	
1970-03-07	1970-01-01	
1970-03-12	1970-01-08	original

## 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.

If query has `FINAL` modifier, the `PREWHERE` optimization is not always correct. It is enabled only if both settings `optimize_move_to_prewhere` and `optimize_move_to_prewhere_if_final` are turned on.

## Attention

The `PREWHERE` section is executed before `FINAL`, so the results of `FROM ... FINAL` queries may be skewed when using `PREWHERE` with fields not in the `ORDER BY` section of a table.

# 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 latency requirements (like below 100ms) but you can't justify the cost of additional hardware resources to meet them.
- When your raw data is not accurate, so approximation does not 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 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 do not know which relative percent of data was processed. So you do not 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 do not 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 Clause

You can use `UNION` with explicitly specifying `UNION ALL` or `UNION DISTINCT`.

If you don't specify `ALL` or `DISTINCT`, it will depend on the `union_default_mode` setting. The difference between `UNION ALL` and `UNION DISTINCT` is that `UNION DISTINCT` will do a distinct transform for union result, it is equivalent to `SELECT DISTINCT` from a subquery containing `UNION ALL`.

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

```
SELECT CounterID, 1 AS table, tolnt64(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` has a `Nullable` type field.

Queries that are parts of `UNION` can 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` in a subquery in the `FROM` clause.

If you use `UNION` without explicitly specifying `UNION ALL` or `UNION DISTINCT`, you can specify the union mode using the `union_default_mode` setting. The setting values can be `ALL`, `DISTINCT` or an empty string. However, if you use `UNION` with `union_default_mode` setting to empty string, it will throw an exception. The following examples demonstrate the results of queries with different values setting.

Query:

```
SET union_default_mode = 'DISTINCT';
SELECT 1 UNION SELECT 2 UNION SELECT 3 UNION SELECT 2;
```

Result:

```
1  
1  
2  
3
```

Query:

```
SET union_default_mode = 'ALL';
SELECT 1 UNION SELECT 2 UNION SELECT 3 UNION SELECT 2;
```

Result:

```
1  
1  
2  
2  
1  
3
```

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

## See Also

- [insert\\_null\\_as\\_default](#) setting.
- [union\\_default\\_mode](#) setting.

[Original article](#)

## 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

ClickHouse supports Common Table Expressions (**CTE**), that is provides to use results of **WITH** clause in the rest of **SELECT** query. Named subqueries can be included to the current and child query context in places where table objects are allowed. Recursion is prevented by hiding the current level CTEs from the **WITH** expression.

## Syntax

```
WITH <expression> AS <identifier>
```

or

```
WITH <identifier> AS <subquery expression>
```

## 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 a sum(bytes) expression result from the 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 a 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: Reusing expression in a subquery

```
WITH test1 AS (SELECT i + 1, j + 1 FROM test1)
SELECT * FROM test1;
```

## INSERT INTO Statement

Adding data.

Basic query format:

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

You can specify a list of columns to insert using the (c1, c2, c3). You can also use an expression with column matcher such as `*` and/or modifiers such as `APPLY`, `EXCEPT`, `REPLACE`.

For example, consider the table:

```
SHOW CREATE insert_select_testtable;
```

```
CREATE TABLE insert_select_testtable
(
    `a` Int8,
    `b` String,
    `c` Int8
)
ENGINE = MergeTree()
ORDER BY a
```

```
INSERT INTO insert_select_testtable (*) VALUES (1, 'a', 1);
```

If you want to insert data in all the columns, except 'b', you need to pass so many values how many columns you chose in parenthesis then:

```
INSERT INTO insert_select_testtable (* EXCEPT(b)) Values (2, 2);
```

```
SELECT * FROM insert_select_testtable;
```

a	b	c
2		2

a	b	c
1	a	1

In this example, we see that the second inserted row has `a` and `c` columns filled by the passed values, and `b` filled with value by default.

If a list of columns does not include all existing columns, the rest of the columns are filled with:

- The values calculated from the `DEFAULT` expressions specified in the table definition.
- Zeros and empty strings, if `DEFAULT` expressions are not defined.

Data can be passed to the `INSERT` in any `format` supported by ClickHouse. The format must be specified explicitly in the query:

```
INSERT INTO [db.]table [(c1, c2, c3)] FORMAT format_name data_set
```

For example, the following query format is identical to the basic version of `INSERT ... VALUES`:

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

ClickHouse removes all spaces and one line feed (if there is one) before the data. When forming a query, we recommend putting the data on a new line after the query operators (this is important if the data begins with spaces).

Example:

```
INSERT INTO t FORMAT TabSeparated
11 Hello, world!
22 Qwerty
```

You can insert data separately from the query by using the command-line client or the HTTP interface. For more information, see the section “[Interfaces](#)”.

## Constraints

If table has [constraints](#), their expressions will be checked for each row of inserted data. If any of those constraints is not satisfied — server will raise an exception containing constraint name and expression, the query will be stopped.

## Inserting the Results of SELECT

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

Columns are mapped according to their position in the `SELECT` clause. However, their names in the `SELECT` expression and the table for `INSERT` may differ. If necessary, type casting is performed.

None of the data formats except `Values` allow setting values to expressions such as `now()`, `1 + 2`, and so on. The `Values` format allows limited use of expressions, but this is not recommended, because in this case inefficient code is used for their execution.

Other queries for modifying data parts are not supported: `UPDATE`, `DELETE`, `REPLACE`, `MERGE`, `UPSERT`, `INSERT UPDATE`.

However, you can delete old data using `ALTER TABLE ... DROP PARTITION`.

`FORMAT` clause must be specified in the end of query if `SELECT` clause contains table function [input\(\)](#).

To insert a default value instead of `NULL` into a column with not nullable data type, enable [insert\\_null\\_as\\_default](#) setting.

## Performance Considerations

`INSERT` sorts the input data by primary key and splits them into partitions by a partition key. If you insert data into several partitions at once, it can significantly reduce the performance of the `INSERT` query. To avoid this:

- Add data in fairly large batches, such as 100,000 rows at a time.
- Group data by a partition key before uploading it to ClickHouse.

Performance will not decrease if:

- Data is added in real time.
- You upload data that is usually sorted by time.

It's also possible to asynchronously insert data in small but frequent inserts. The data from such insertions is combined into batches and then safely inserted into a table. To enable the asynchronous mode, switch on the [async\\_insert](#) setting. Note that asynchronous insertions are supported only over HTTP protocol, and deduplication is not supported for them.

## See Also

- [async\\_insert](#)
- [async\\_insert\\_threads](#)
- [wait\\_for\\_async\\_insert](#)
- [wait\\_for\\_async\\_insert\\_timeout](#)
- [async\\_insert\\_max\\_data\\_size](#)
- [async\\_insert\\_busy\\_timeout\\_ms](#)
- [async\\_insert\\_stale\\_timeout\\_ms](#)

---

# CREATE Queries

Create queries make a new entity of one of the following kinds:

- [DATABASE](#)
- [TABLE](#)
- [VIEW](#)
- [DICTIONARY](#)
- [FUNCTION](#)
- [USER](#)
- [ROLE](#)
- [ROW POLICY](#)
- [QUOTA](#)
- [SETTINGS PROFILE](#)

---

## CREATE DATABASE

Creates a new database.

```
CREATE DATABASE [IF NOT EXISTS] db_name [ON CLUSTER cluster] [ENGINE = engine(...)]
```

## Clauses

### IF NOT EXISTS

If the `db_name` database already exists, then ClickHouse does not create a new database and:

- Doesn't throw an exception if clause is specified.
- Throws an exception if clause isn't specified.

## ON CLUSTER

ClickHouse creates the `db_name` database on all the servers of a specified cluster. More details in a [Distributed DDL](#) article.

## ENGINE

[MySQL](#) allows you to retrieve data from the remote MySQL server. By default, ClickHouse uses its own [database engine](#). There's also a [lazy](#) engine.

# CREATE TABLE

Creates a new table. This query can have various syntax forms depending on a use case.

By default, tables are created only on the current server. Distributed DDL queries are implemented as `ON CLUSTER` clause, which is [described separately](#).

## Syntax Forms

### With Explicit Schema

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
(
    name1 [type1] [NULL|NOT NULL] [DEFAULT|MATERIALIZED|ALIAS expr1] [compression_codec] [TTL expr1],
    name2 [type2] [NULL|NOT NULL] [DEFAULT|MATERIALIZED|ALIAS expr2] [compression_codec] [TTL expr2],
    ...
) ENGINE = engine
```

Creates a table named `name` in the `db` database or the current database if `db` is not set, with the structure specified in brackets and the `engine` engine.

The structure of the table is a list of column descriptions, secondary indexes and constraints . If [primary key](#) is supported by the engine, it will be indicated as parameter for the table engine.

A column description is `name type` in the simplest case. Example: `RegionID UInt32`.

Expressions can also be defined for default values (see below).

If necessary, primary key can be specified, with one or more key expressions.

### With a Schema Similar to Other Table

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name AS [db2.]name2 [ENGINE = engine]
```

Creates a table with the same structure as another table. You can specify a different engine for the table. If the engine is not specified, the same engine will be used as for the `db2.name2` table.

### From a Table Function

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name AS table_function()
```

Creates a table with the same result as that of the `table function` specified. The created table will also work in the same way as the corresponding table function that was specified.

## From SELECT query

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name[(name1 [type1], name2 [type2], ...)] ENGINE = engine AS SELECT ...
```

Creates a table with a structure like the result of the `SELECT` query, with the `engine` engine, and fills it with data from `SELECT`. Also you can explicitly specify columns description.

If the table already exists and `IF NOT EXISTS` is specified, the query won't do anything.

There can be other clauses after the `ENGINE` clause in the query. See detailed documentation on how to create tables in the descriptions of `table engines`.

### Example

Query:

```
CREATE TABLE t1 (x String) ENGINE = Memory AS SELECT 1;
SELECT x, toTypeName(x) FROM t1;
```

Result:

x	toTypeName(x)
1	String

## NULL Or NOT NULL Modifiers

`NULL` and `NOT NULL` modifiers after data type in column definition allow or do not allow it to be `Nullable`.

If the type is not `Nullable` and if `NULL` is specified, it will be treated as `Nullable`; if `NOT NULL` is specified, then no. For example, `INT NULL` is the same as `Nullable(INT)`. If the type is `Nullable` and `NULL` or `NOT NULL` modifiers are specified, the exception will be thrown.

See also [data\\_type\\_default\\_nullable](#) setting.

## Default Values

The column description can specify an expression for a default value, in one of the following ways: `DEFAULT expr`, `MATERIALIZED expr`, `ALIAS expr`.

Example: `URLDomain String DEFAULT domain(URL)`.

If an expression for the default value is not defined, the default values will be set to zeros for numbers, empty strings for strings, empty arrays for arrays, and `1970-01-01` for dates or zero unix timestamp for `DateTime`, `NULL` for `Nullable`.

If the default expression is defined, the column type is optional. If there isn't an explicitly defined type, the default expression type is used. Example: `EventDate DEFAULT toDate(EventTime)` – the 'Date' type will be used for the 'EventDate' column.

If the data type and default expression are defined explicitly, this expression will be cast to the specified type using type casting functions. Example: `Hits UInt32 DEFAULT 0` means the same thing as `Hits UInt32 DEFAULT toUInt32(0)`.

Default expressions may be defined as an arbitrary expression from table constants and columns. When creating and changing the table structure, it checks that expressions do not contain loops. For INSERT, it checks that expressions are resolvable – that all columns they can be calculated from have been passed.

## DEFAULT

### DEFAULT expr

Normal default value. If the INSERT query does not specify the corresponding column, it will be filled in by computing the corresponding expression.

## MATERIALIZED

### MATERIALIZED expr

Materialized expression. Such a column can't be specified for INSERT, because it is always calculated. For an INSERT without a list of columns, these columns are not considered. In addition, this column is not substituted when using an asterisk in a SELECT query. This is to preserve the invariant that the dump obtained using `SELECT *` can be inserted back into the table using INSERT without specifying the list of columns.

## ALIAS

### ALIAS expr

Synonym. Such a column isn't stored in the table at all.

Its values can't be inserted in a table, and it is not substituted when using an asterisk in a SELECT query. It can be used in SELECTs if the alias is expanded during query parsing.

When using the ALTER query to add new columns, old data for these columns is not written. Instead, when reading old data that does not have values for the new columns, expressions are computed on the fly by default. However, if running the expressions requires different columns that are not indicated in the query, these columns will additionally be read, but only for the blocks of data that need it.

If you add a new column to a table but later change its default expression, the values used for old data will change (for data where values were not stored on the disk). Note that when running background merges, data for columns that are missing in one of the merging parts is written to the merged part.

It is not possible to set default values for elements in nested data structures.

## Primary Key

You can define a **primary key** when creating a table. Primary key can be specified in two ways:

- Inside the column list

```
CREATE TABLE db.table_name
(
    name1 type1, name2 type2, ...,
    PRIMARY KEY(expr1[, expr2,...])
)
ENGINE = engine;
```

- Outside the column list

```
CREATE TABLE db.table_name
(
    name1 type1, name2 type2, ...
)
ENGINE = engine
PRIMARY KEY(expr1[, expr2,...]);
```

## Warning

You can't combine both ways in one query.

## Constraints

Along with columns descriptions constraints could be defined:

```
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` could by any boolean expression. If constraints are defined for the table, each of them will be checked for every row in `INSERT` query. If any constraint is not satisfied — server will raise an exception with constraint name and checking expression.

Adding large amount of constraints can negatively affect performance of big `INSERT` queries.

## TTL Expression

Defines storage time for values. Can be specified only for MergeTree-family tables. For the detailed description, see [TTL for columns and tables](#).

## Column Compression Codecs

By default, ClickHouse applies the `Iz4` compression method. For MergeTree-engine family you can change the default compression method in the `compression` section of a server configuration.

You can also define the compression method for each individual column in the `CREATE TABLE` query.

```
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>
...
```

The `Default` codec can be specified to reference default compression which may depend on different settings (and properties of data) in runtime.

Example: `value UInt64 CODEC(Default)` — the same as lack of codec specification.

Also you can remove current CODEC from the column and use default compression from config.xml:

```
ALTER TABLE codec_example MODIFY COLUMN float_value CODEC(Default);
```

Codecs can be combined in a pipeline, for example, `CODEC(Delta, Default)`.

## Warning

You can't decompress ClickHouse database files with external utilities like `lz4`. Instead, use the special **clickhouse-compressor** utility.

Compression is supported for the following table engines:

- **MergeTree** family. Supports column compression codecs and selecting the default compression method by `compression` settings.
- **Log** family. Uses the `lz4` compression method by default and supports column compression codecs.
- **Set**. Only supported the default compression.
- **Join**. Only supported the default compression.

ClickHouse supports general purpose codecs and specialized codecs.

## General Purpose Codecs

Codecs:

- `NONE` — No compression.
- `LZ4` — Lossless **data compression algorithm** used by default. Applies LZ4 fast compression.
- `LZ4HC[(level)]` — LZ4 HC (high compression) algorithm with configurable level. Default level: 9. Setting `level <= 0` applies the default level. Possible levels: [1, 12]. Recommended level range: [4, 9].
- `ZSTD[(level)]` — **ZSTD compression algorithm** with configurable `level`. Possible levels: [1, 22]. Default value: 1.

High compression levels are useful for asymmetric scenarios, like compress once, decompress repeatedly. Higher levels mean better compression and higher CPU usage.

## Specialized Codecs

These codecs are designed to make compression more effective by using specific features of data. Some of these codecs do not compress data themselves. Instead, they prepare the data for a common purpose codec, which compresses it better than without this preparation.

Specialized codecs:

- `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` are used for storing delta values, so `delta_bytes` is the maximum size of raw values. Possible `delta_bytes` values: 1, 2, 4, 8. The default value for `delta_bytes` is `sizeof(type)` if equal to 1, 2, 4, or 8. In all other cases, it's 1.

- **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: A Fast, Scalable, In-Memory Time Series Database](#).
- **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: A Fast, Scalable, In-Memory Time Series Database](#).
- **T64** — Compression approach that crops unused high bits of values in integer data types (including `Enum`, `Date` and `DateTime`). At each step of its algorithm, codec takes a block of 64 values, puts them into 64x64 bit matrix, transposes it, crops the unused bits of values and returns the rest as a sequence. Unused bits are the bits, that do not differ between maximum and minimum values in the whole data part for which the compression is used.

`DoubleDelta` and `Gorilla` codecs are used in Gorilla TSDB as the components of its compressing algorithm. Gorilla approach is effective in scenarios when there is a sequence of slowly changing values with their timestamps. Timestamps are effectively compressed by the `DoubleDelta` codec, and values are effectively compressed by the `Gorilla` codec. For example, to get an effectively stored table, you can create it in the following configuration:

```
CREATE TABLE codec_example
(
    timestamp DateTime CODEC(DoubleDelta),
    slow_values Float32 CODEC(Gorilla)
)
ENGINE = MergeTree()
```

## Temporary Tables

ClickHouse supports temporary tables which have the following characteristics:

- Temporary tables disappear when the session ends, including if the connection is lost.
- A temporary table uses the Memory engine only.
- The DB can't be specified for a temporary table. It is created outside of databases.
- Impossible to create a temporary table with distributed DDL query on all cluster servers (by using `ON CLUSTER`): this table exists only in the current session.
- If a temporary table has the same name as another one and a query specifies the table name without specifying the DB, the temporary table will be used.
- For distributed query processing, temporary tables used in a query are passed to remote servers.

To create a temporary table, use the following syntax:

```
CREATE TEMPORARY TABLE [IF NOT EXISTS] table_name
(
    name1 [type1][DEFAULT|MATERIALIZED|ALIAS expr1],
    name2 [type2][DEFAULT|MATERIALIZED|ALIAS expr2],
    ...
)
```

In most cases, temporary tables are not created manually, but when using external data for a query, or for distributed (GLOBAL) IN. For more information, see the appropriate sections

It's possible to use tables with **ENGINE = Memory** instead of temporary tables.

## REPLACE TABLE

'REPLACE' query allows you to update the table atomically.

### Note

This query is supported only for **Atomic** database engine.

If you need to delete some data from a table, you can create a new table and fill it with a **SELECT** statement that does not retrieve unwanted data, then drop the old table and rename the new one:

```
CREATE TABLE myNewTable AS myOldTable;
INSERT INTO myNewTable SELECT * FROM myOldTable WHERE CounterID <12345;
DROP TABLE myOldTable;
RENAME TABLE myNewTable TO myOldTable;
```

Instead of above, you can use the following:

```
REPLACE TABLE myOldTable SELECT * FROM myOldTable WHERE CounterID <12345;
```

## Syntax

```
{CREATE [OR REPLACE] | REPLACE} TABLE [db.]table_name
```

All syntax forms for **CREATE** query also work for this query. **REPLACE** for a non-existent table will cause an error.

## Examples:

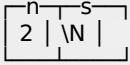
Consider the table:

```
CREATE DATABASE base ENGINE = Atomic;
CREATE OR REPLACE TABLE base.t1 (n UInt64, s String) ENGINE = MergeTree ORDER BY n;
INSERT INTO base.t1 VALUES (1, 'test');
SELECT * FROM base.t1;
```

n	s
1	test

Using **REPLACE** query to clear all data:

```
CREATE OR REPLACE TABLE base.t1 (n UInt64, s Nullable(String)) ENGINE = MergeTree ORDER BY n;
INSERT INTO base.t1 VALUES (2, null);
SELECT * FROM base.t1;
```



Using `REPLACE` query to change table structure:

```
REPLACE TABLE base.t1 (n UInt64) ENGINE = MergeTree ORDER BY n;
INSERT INTO base.t1 VALUES (3);
SELECT * FROM base.t1;
```



## COMMENT Clause

You can add a comment to the table when you creating it.

### Note

The comment is supported for all table engines except **Kafka**, **RabbitMQ** and **EmbeddedRocksDB**.

### Syntax

```
CREATE TABLE db.table_name
(
    name1 type1, name2 type2, ...
)
ENGINE = engine
COMMENT 'Comment'
```

### Example

Query:

```
CREATE TABLE t1 (x String) ENGINE = Memory COMMENT 'The temporary table';
SELECT name, comment FROM system.tables WHERE name = 't1';
```

Result:

name	comment
t1	The temporary table

## CREATE VIEW

Creates a new view. Views can be **normal**, **materialized** and **live** (the latter is an experimental feature).

### Normal View

Syntax:

```
CREATE [OR REPLACE] VIEW [IF NOT EXISTS] [db.]table_name [ON CLUSTER] AS SELECT ...
```

Normal views do not store any data. They just perform a read from another table on each access. In other words, a normal view is nothing more than a saved query. When reading from a view, this saved query is used as a subquery in the **FROM** clause.

As an example, assume you've created a view:

```
CREATE VIEW view AS SELECT ...
```

and written a query:

```
SELECT a, b, c FROM view
```

This query is fully equivalent to using the subquery:

```
SELECT a, b, c FROM (SELECT ...)
```

## Materialized View

```
CREATE MATERIALIZED VIEW [IF NOT EXISTS] [db.]table_name [ON CLUSTER] [TO[db.]name] [ENGINE = engine]  
[POPULATE] AS SELECT ...
```

Materialized views store data transformed by the corresponding **SELECT** query.

When creating a materialized view without **TO [db].[table]**, you must specify **ENGINE** – the table engine for storing data.

When creating a materialized view with **TO [db].[table]**, you must not use **POPULATE**.

A materialized view is implemented as follows: when inserting data to the table specified in **SELECT**, part of the inserted data is converted by this **SELECT** query, and the result is inserted in the view.

## Important

Materialized views in ClickHouse use **column names** instead of column order during insertion into destination table. If some column names are not present in the **SELECT** query result, ClickHouse uses a default value, even if the column is not **Nullable**. A safe practice would be to add aliases for every column when using Materialized views.

Materialized views in ClickHouse are implemented more like insert triggers. If there's some aggregation in the view query, it's applied only to the batch of freshly inserted data. Any changes to existing data of source table (like update, delete, drop partition, etc.) does not change the materialized view.

If you specify **POPULATE**, the existing table data is inserted into the view when creating it, as if making a **CREATE TABLE ... AS SELECT ...**. Otherwise, the query contains only the data inserted in the table after creating the view. We **do not recommend** using **POPULATE**, since data inserted in the table during the view creation will not be inserted in it.

A `SELECT` query can contain `DISTINCT`, `GROUP BY`, `ORDER BY`, `LIMIT`. Note that the corresponding conversions are performed independently on each block of inserted data. For example, if `GROUP BY` is set, data is aggregated during insertion, but only within a single packet of inserted data. The data won't be further aggregated. The exception is when using an `ENGINE` that independently performs data aggregation, such as `SummingMergeTree`.

The execution of `ALTER` queries on materialized views has limitations, so they might be inconvenient. If the materialized view uses the construction `TO [db.]name`, you can `DETACH` the view, run `ALTER` for the target table, and then `ATTACH` the previously detached (`DETACH`) view.

Note that materialized view is influenced by `optimize_on_insert` setting. The data is merged before the insertion into a view.

Views look the same as normal tables. For example, they are listed in the result of the `SHOW TABLES` query.

To delete a view, use `DROP VIEW`. Although `DROP TABLE` works for VIEWS as well.

## Live View [Experimental]

### Important

This is an experimental feature that may change in backwards-incompatible ways in the future releases.

Enable usage of live views and `WATCH` query using `allow_experimental_live_view` setting. Input the command `set allow_experimental_live_view = 1`.

```
CREATE LIVE VIEW [IF NOT EXISTS] [db.]table_name [WITH [TIMEOUT [value_in_sec] [AND]] [REFRESH [value_in_sec]]]
AS SELECT ...
```

Live views store result of the corresponding `SELECT` query and are updated any time the result of the query changes. Query result as well as partial result needed to combine with new data are stored in memory providing increased performance for repeated queries. Live views can provide push notifications when query result changes using the `WATCH` query.

Live views are triggered by insert into the innermost table specified in the query.

Live views work similarly to how a query in a distributed table works. But instead of combining partial results from different servers they combine partial result from current data with partial result from the new data. When a live view query includes a subquery then the cached partial result is only stored for the innermost subquery.

## Limitations

- **Table function** is not supported as the innermost table.
- Tables that do not have inserts such as a **dictionary**, **system table**, a **normal view**, or a **materialized view** will not trigger a live view.
- Only queries where one can combine partial result from the old data plus partial result from the new data will work. Live view will not work for queries that require the complete data set to compute the final result or aggregations where the state of the aggregation must be preserved.
- Does not work with replicated or distributed tables where inserts are performed on different nodes.
- Can't be triggered by multiple tables.

See **WITH REFRESH** to force periodic updates of a live view that in some cases can be used as a workaround.

## Monitoring Live View Changes

You can monitor changes in the `LIVE VIEW` query result using **WATCH** query.

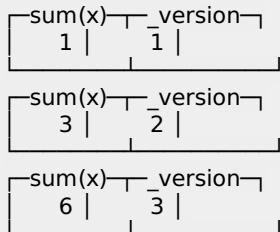
```
WATCH [db.]live_view
```

### Example:

```
CREATE TABLE mt (x Int8) Engine = MergeTree ORDER BY x;
CREATE LIVE VIEW lv AS SELECT sum(x) FROM mt;
```

Watch a live view while doing a parallel insert into the source table.

```
WATCH lv;
```



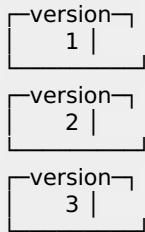
```
INSERT INTO mt VALUES (1);
INSERT INTO mt VALUES (2);
INSERT INTO mt VALUES (3);
```

Or add **EVENTS** clause to just get change events.

```
WATCH [db.]live_view EVENTS;
```

### Example:

```
WATCH lv EVENTS;
```



You can execute **SELECT** query on a live view in the same way as for any regular view or a table. If the query result is cached it will return the result immediately without running the stored query on the underlying tables.

```
SELECT * FROM [db.]live_view WHERE ...
```

## Force Live View Refresh

You can force live view refresh using the `ALTER LIVE VIEW [db.]table_name REFRESH` statement.

### WITH TIMEOUT Clause

When a live view is created with a `WITH TIMEOUT` clause then the live view will be dropped automatically after the specified number of seconds elapse since the end of the last `WATCH` query that was watching the live view.

```
CREATE LIVE VIEW [db.]table_name WITH TIMEOUT [value_in_sec] AS SELECT ...
```

If the timeout value is not specified then the value specified by the `temporary_live_view_timeout` setting is used.

#### Example:

```
CREATE TABLE mt (x Int8) Engine = MergeTree ORDER BY x;
CREATE LIVE VIEW lv WITH TIMEOUT 15 AS SELECT sum(x) FROM mt;
```

### WITH REFRESH Clause

When a live view is created with a `WITH REFRESH` clause then it will be automatically refreshed after the specified number of seconds elapse since the last refresh or trigger.

```
CREATE LIVE VIEW [db.]table_name WITH REFRESH [value_in_sec] AS SELECT ...
```

If the refresh value is not specified then the value specified by the `periodic_live_view_refresh` setting is used.

#### Example:

```
CREATE LIVE VIEW lv WITH REFRESH 5 AS SELECT now();
WATCH lv
```

now()	version
2021-02-21 08:47:05	1
2021-02-21 08:47:10	2
2021-02-21 08:47:15	3

You can combine `WITH TIMEOUT` and `WITH REFRESH` clauses using an `AND` clause.

```
CREATE LIVE VIEW [db.]table_name WITH TIMEOUT [value_in_sec] AND REFRESH [value_in_sec] AS SELECT ...
```

#### Example:

```
CREATE LIVE VIEW lv WITH TIMEOUT 15 AND REFRESH 5 AS SELECT now();
```

After 15 sec the live view will be automatically dropped if there are no active `WATCH` queries.

```
WATCH lv
```

```
Code: 60. DB::Exception: Received from localhost:9000. DB::Exception: Table default.lv does not exist..
```

## Live View Usage

Most common uses of live view tables include:

- Providing push notifications for query result changes to avoid polling.
- Caching results of most frequent queries to provide immediate query results.
- Watching for table changes and triggering a follow-up select queries.
- Watching metrics from system tables using periodic refresh.

### See Also

- [ALTER LIVE VIEW](#)

## CREATE DICTIONARY

Creates a new [external dictionary](#) with given [structure](#), [source](#), [layout](#) and [lifetime](#).

Syntax:

```
CREATE DICTIONARY [IF NOT EXISTS] [db.]dictionary_name [ON CLUSTER cluster]
(
    key1 type1 [DEFAULT|EXPRESSION expr1] [IS_OBJECT_ID],
    key2 type2 [DEFAULT|EXPRESSION expr2],
    attr1 type2 [DEFAULT|EXPRESSION expr3] [HIERARCHICAL|INJECTIVE],
    attr2 type2 [DEFAULT|EXPRESSION expr4] [HIERARCHICAL|INJECTIVE]
)
PRIMARY KEY key1, key2
SOURCE(SOURCE_NAME([param1 value1 ... paramN valueN]))
LAYOUT(LAYOUT_NAME([param_name param_value]))
LIFETIME({MIN min_val MAX max_val | max_val})
```

External dictionary structure consists of attributes. Dictionary attributes are specified similarly to table columns. The only required attribute property is its type, all other properties may have default values.

`ON CLUSTER` clause allows creating dictionary on a cluster, see [Distributed DDL](#).

Depending on dictionary [layout](#) one or more attributes can be specified as dictionary keys.

For more information, see [External Dictionaries](#) section.

## CREATE FUNCTION

Creates a user defined function from a lambda expression. The expression must consist of function parameters, constants, operators, or other function calls.

### Syntax

```
CREATE FUNCTION name AS (parameter0, ...) -> expression
```

A function can have an arbitrary number of parameters.

There are a few restrictions:

- The name of a function must be unique among user defined and system functions.
- Recursive functions are not allowed.
- All variables used by a function must be specified in its parameter list.

If any restriction is violated then an exception is raised.

### Example

Query:

```
CREATE FUNCTION linear_equation AS (x, k, b) -> k*x + b;
SELECT number, linear_equation(number, 2, 1) FROM numbers(3);
```

Result:

number	linear_equation(number, 2, 1)
0	0
1	2
2	4

A **conditional function** is called in a user defined function in the following query:

```
CREATE FUNCTION parity_str AS (n) -> if(n % 2, 'odd', 'even');
SELECT number, parity_str(number) FROM numbers(3);
```

Result:

number	parity_str(number)
0	even
1	odd
2	even

## CREATE USER

Creates **user accounts**.

Syntax:

```
CREATE USER [IF NOT EXISTS | OR REPLACE] name1 [ON CLUSTER cluster_name1]
[, name2 [ON CLUSTER cluster_name2] ...]
[NOT IDENTIFIED | IDENTIFIED {[WITH {no_password | plaintext_password | sha256_password | sha256_hash |
double_sha1_password | double_sha1_hash} BY {'password' | 'hash'} } | {WITH Idap SERVER 'server_name'} | {WITH
kerberos [REALM 'realm']} ]
[HOST {LOCAL | NAME 'name' | REGEXP 'name_regexp' | IP 'address' | LIKE 'pattern'} [...]
| ANY | NONE]
[DEFAULT ROLE role [...]]
[DEFAULT DATABASE database | NONE]
[GRANTEES {user | role | ANY | NONE} [...] [EXCEPT {user | role} [...]]]
[SETTINGS variable [= value] [MIN [=] min_value] [MAX [=] max_value] [READONLY | WRITABLE] | PROFILE
'profile_name'] [...]
```

ON CLUSTER clause allows creating users on a cluster, see [Distributed DDL](#).

# Identification

There are multiple ways of user identification:

- IDENTIFIED WITH no\_password
- IDENTIFIED WITH plaintext\_password BY 'qwerty'
- IDENTIFIED WITH sha256\_password BY 'qwerty' or IDENTIFIED BY 'password'
- IDENTIFIED WITH sha256\_hash BY 'hash'
- IDENTIFIED WITH double\_sha1\_password BY 'qwerty'
- IDENTIFIED WITH double\_sha1\_hash BY 'hash'
- IDENTIFIED WITH ldap SERVER 'server\_name'
- IDENTIFIED WITH kerberos or IDENTIFIED WITH kerberos REALM 'realm'

## User Host

User host is a host from which a connection to ClickHouse server could be established. The host can be specified in the `HOST` query section in the following ways:

- `HOST IP 'ip_address_or_subnetwork'` — User can connect to ClickHouse server only from the specified IP address or a **subnetwork**. Examples: `HOST IP '192.168.0.0/16'`, `HOST IP '2001:DB8::/32'`. For use in production, only specify `HOST IP` elements (IP addresses and their masks), since using `host` and `host_regexp` might cause extra latency.
- `HOST ANY` — User can connect from any location. This is a default option.
- `HOST LOCAL` — User can connect only locally.
- `HOST NAME 'fqdn'` — User host can be specified as FQDN. For example, `HOST NAME 'mysite.com'`.
- `HOST NAME REGEXP 'regexp'` — You can use **pcre** regular expressions when specifying user hosts. For example, `HOST NAME REGEXP '.*\.mysite\.com'`.
- `HOST LIKE 'template'` — Allows you to use the **LIKE** operator to filter the user hosts. For example, `HOST LIKE '%'` is equivalent to `HOST ANY`, `HOST LIKE '%.mysite.com'` filters all the hosts in the `mysite.com` domain.

Another way of specifying host is to use `@` syntax following the username. Examples:

- `CREATE USER mira@'127.0.0.1'` — Equivalent to the `HOST IP` syntax.
- `CREATE USER mira@'localhost'` — Equivalent to the `HOST LOCAL` syntax.
- `CREATE USER mira@'192.168.%.%'` — Equivalent to the `HOST LIKE` syntax.

### Warning

ClickHouse treats `user_name@'address'` as a username as a whole. Thus, technically you can create multiple users with the same `user_name` and different constructions after `@`. However, we do not recommend to do so.

## GRANTEES Clause

Specifies users or roles which are allowed to receive **privileges** from this user on the condition this user has also all required access granted with **GRANT OPTION**. Options of the **GRANTEES** clause:

- **user** — Specifies a user this user can grant privileges to.
- **role** — Specifies a role this user can grant privileges to.
- **ANY** — This user can grant privileges to anyone. It's the default setting.
- **NONE** — This user can grant privileges to none.

You can exclude any user or role by using the **EXCEPT** expression. For example, `CREATE USER user1 GRANTEES ANY EXCEPT user2`. It means if `user1` has some privileges granted with **GRANT OPTION** it will be able to grant those privileges to anyone except `user2`.

## Examples

Create the user account `mira` protected by the password `qwerty`:

```
CREATE USER mira HOST IP '127.0.0.1' IDENTIFIED WITH sha256_password BY 'qwerty';
```

`mira` should start client app at the host where the ClickHouse server runs.

Create the user account `john`, assign roles to it and make this roles default:

```
CREATE USER john DEFAULT ROLE role1, role2;
```

Create the user account `john` and make all his future roles default:

```
CREATE USER john DEFAULT ROLE ALL;
```

When some role is assigned to `john` in the future, it will become default automatically.

Create the user account `john` and make all his future roles default excepting `role1` and `role2`:

```
CREATE USER john DEFAULT ROLE ALL EXCEPT role1, role2;
```

Create the user account `john` and allow him to grant his privileges to the user with `jack` account:

```
CREATE USER john GRANTEES jack;
```

## CREATE ROLE

Creates new **roles**. Role is a set of **privileges**. A **user** assigned a role gets all the privileges of this role.

Syntax:

```
CREATE ROLE [IF NOT EXISTS | OR REPLACE] name1 [, name2 ...]  
[SETTINGS variable [= value] [MIN [=] min_value] [MAX [=] max_value] [READONLY|WRITABLE] | PROFILE  
'profile_name'] [...]
```

## Managing Roles

A user can be assigned multiple roles. Users can apply their assigned roles in arbitrary combinations by the **SET ROLE** statement. The final scope of privileges is a combined set of all the privileges of all the applied roles. If a user has privileges granted directly to its user account, they are also combined with the privileges granted by roles.

User can have default roles which apply at user login. To set default roles, use the **SET DEFAULT ROLE** statement or the **ALTER USER** statement.

To revoke a role, use the **REVOKE** statement.

To delete role, use the **DROP ROLE** statement. The deleted role is being automatically revoked from all the users and roles to which it was assigned.

## Examples

```
CREATE ROLE accountant;
GRANT SELECT ON db.* TO accountant;
```

This sequence of queries creates the role **accountant** that has the privilege of reading data from the **accounting** database.

Assigning the role to the user **mira**:

```
GRANT accountant TO mira;
```

After the role is assigned, the user can apply it and execute the allowed queries. For example:

```
SET ROLE accountant;
SELECT * FROM db.*;
```

## CREATE ROW POLICY

Creates a **row policy**, i.e. a filter used to determine which rows a user can read from a table.

### Warning

Row policies make sense only for users with readonly access. If user can modify table or copy partitions between tables, it defeats the restrictions of row policies.

Syntax:

```
CREATE [ROW] POLICY [IF NOT EXISTS | OR REPLACE] policy_name1 [ON CLUSTER cluster_name1] ON [db1.]table1
[, policy_name2 [ON CLUSTER cluster_name2] ON [db2.]table2 ...]
[FOR SELECT] USING condition
[AS {PERMISSIVE | RESTRICTIVE}]
[TO {role1 [, role2 ...] | ALL | ALL EXCEPT role1 [, role2 ...]}]
```

## USING Clause

Allows to specify a condition to filter rows. An user will see a row if the condition is calculated to non-zero for the row.

## TO Clause

In the section `TO` you can provide a list of users and roles this policy should work for. For example, `CREATE ROW POLICY ... TO accountant, john@localhost`.

Keyword `ALL` means all the ClickHouse users including current user. Keyword `ALL EXCEPT` allow to exclude some users from the all users list, for example, `CREATE ROW POLICY ... TO ALL EXCEPT accountant, john@localhost`

## Note

If there are no row policies defined for a table then any user can `SELECT` all the rows from the table. Defining one or more row policies for the table makes the access to the table depending on the row policies no matter if those row policies are defined for the current user or not. For example, the following policy

```
CREATE ROW POLICY pol1 ON mydb.table1 USING b=1 TO mira, peter
```

forbids the users `mira` and `peter` to see the rows with `b != 1`, and any non-mentioned user (e.g., the user `paul`) will see no rows from `mydb.table1` at all.

If that's not desirable it can't be fixed by adding one more row policy, like the following:

```
CREATE ROW POLICY pol2 ON mydb.table1 USING 1 TO ALL EXCEPT mira, peter
```

## AS Clause

It's allowed to have more than one policy enabled on the same table for the same user at the one time. So we need a way to combine the conditions from multiple policies.

By default policies are combined using the boolean `OR` operator. For example, the following policies

```
CREATE ROW POLICY pol1 ON mydb.table1 USING b=1 TO mira, peter  
CREATE ROW POLICY pol2 ON mydb.table1 USING c=2 TO peter, antonio
```

enables the user `peter` to see rows with either `b=1` or `c=2`.

The `AS` clause specifies how policies should be combined with other policies. Policies can be either permissive or restrictive. By default policies are permissive, which means they are combined using the boolean `OR` operator.

A policy can be defined as restrictive as an alternative. Restrictive policies are combined using the boolean `AND` operator.

Here is the general formula:

```
row_is_visible = (one or more of the permissive policies' conditions are non-zero) AND  
(all of the restrictive policies's conditions are non-zero)
```

For example, the following policies

```
CREATE ROW POLICY pol1 ON mydb.table1 USING b=1 TO mira, peter  
CREATE ROW POLICY pol2 ON mydb.table1 USING c=2 AS RESTRICTIVE TO peter, antonio
```

enables the user `peter` to see rows only if both `b=1` AND `c=2`.

## ON CLUSTER Clause

Allows creating row policies on a cluster, see [Distributed DDL](#).

# Examples

```
CREATE ROW POLICY filter1 ON mydb.mytable USING a<1000 TO accountant, john@localhost
```

```
CREATE ROW POLICY filter2 ON mydb.mytable USING a<1000 AND b=5 TO ALL EXCEPT mira
```

```
CREATE ROW POLICY filter3 ON mydb.mytable USING 1 TO admin
```

## CREATE QUOTA

Creates a **quota** that can be assigned to a user or a role.

Syntax:

```
CREATE QUOTA [IF NOT EXISTS | OR REPLACE] name [ON CLUSTER cluster_name]
    [KEYED BY {user_name | ip_address | client_key | client_key,user_name | client_key,ip_address} | NOT KEYED]
    [FOR [RANDOMIZED] INTERVAL number {second | minute | hour | day | week | month | quarter | year}
        {MAX { {queries | query_selects | query_inserts | errors | result_rows | result_bytes | read_rows | read_bytes | execution_time} = number } [...] | NO LIMITS | TRACKING ONLY} [...] ]
    [TO {role [...] | ALL | ALL EXCEPT role [...]}]
```

Keys `user_name`, `ip_address`, `client_key`, `client_key, user_name` and `client_key, ip_address` correspond to the fields in the **system.quotas** table.

Parameters `queries`, `query_selects`, `query_inserts`, `errors`, `result_rows`, `result_bytes`, `read_rows`, `read_bytes`, `execution_time` correspond to the fields in the **system.quotas\_usage** table.

`ON CLUSTER` clause allows creating quotas on a cluster, see [Distributed DDL](#).

### Examples

Limit the maximum number of queries for the current user with 123 queries in 15 months constraint:

```
CREATE QUOTA qA FOR INTERVAL 15 month MAX queries = 123 TO CURRENT_USER;
```

For the default user limit the maximum execution time with half a second in 30 minutes, and limit the maximum number of queries with 321 and the maximum number of errors with 10 in 5 quarters:

```
CREATE QUOTA qB FOR INTERVAL 30 minute MAX execution_time = 0.5, FOR INTERVAL 5 quarter MAX queries = 321,
errors = 10 TO default;
```

## CREATE SETTINGS PROFILE

Creates **settings profiles** that can be assigned to a user or a role.

Syntax:

```
CREATE SETTINGS PROFILE [IF NOT EXISTS | OR REPLACE] TO name1 [ON CLUSTER cluster_name1]
    [, name2 [ON CLUSTER cluster_name2] ...]
    [SETTINGS variable [= value] [MIN [=] min_value] [MAX [=] max_value] [READONLY|WRITABLE] | INHERIT
    'profile_name'] [...]
```

`ON CLUSTER` clause allows creating settings profiles on a cluster, see [Distributed DDL](#).

## Example

Create the `max_memory_usage_profile` settings profile with value and constraints for the `max_memory_usage` setting and assign it to user `robin`:

```
CREATE SETTINGS PROFILE max_memory_usage_profile SETTINGS max_memory_usage = 100000001 MIN 90000000  
MAX 110000000 TO robin
```

## ALTER

Most `ALTER TABLE` queries modify table settings or data:

- [COLUMN](#)
- [PARTITION](#)
- [DELETE](#)
- [UPDATE](#)
- [ORDER BY](#)
- [INDEX](#)
- [CONSTRAINT](#)
- [TTL](#)

### Note

Most `ALTER TABLE` queries are supported only for **\*MergeTree** tables, as well as **Merge** and **Distributed**.

These `ALTER` statements manipulate views:

- [ALTER TABLE ... MODIFY QUERY](#) — Modifies a [Materialized view](#) structure.
- [ALTER LIVE VIEW](#) — Refreshes a [Live view](#).

These `ALTER` statements modify entities related to role-based access control:

- [USER](#)
- [ROLE](#)
- [QUOTA](#)
- [ROW POLICY](#)
- [SETTINGS PROFILE](#)

## Mutations

`ALTER` queries that are intended to manipulate table data are implemented with a mechanism called “mutations”, most notably [ALTER TABLE ... DELETE](#) and [ALTER TABLE ... UPDATE](#). They are asynchronous background processes similar to merges in **MergeTree** tables that produce new “mutated” versions of parts.

For `*MergeTree` tables mutations execute by **rewriting whole data parts**. There is no atomicity - parts are substituted for mutated parts as soon as they are ready and a `SELECT` query that started executing during a mutation will see data from parts that have already been mutated along with data from parts that have not been mutated yet.

Mutations are totally ordered by their creation order and are applied to each part in that order. Mutations are also partially ordered with `INSERT INTO` queries: data that was inserted into the table before the mutation was submitted will be mutated and data that was inserted after that will not be mutated. Note that mutations do not block inserts in any way.

A mutation query returns immediately after the mutation entry is added (in case of replicated tables to ZooKeeper, for non-replicated tables - to the filesystem). The mutation itself executes asynchronously using the system profile settings. To track the progress of mutations you can use the `system.mutations` table. A mutation that was successfully submitted will continue to execute even if ClickHouse servers are restarted. There is no way to roll back the mutation once it is submitted, but if the mutation is stuck for some reason it can be cancelled with the `KILL MUTATION` query.

Entries for finished mutations are not deleted right away (the number of preserved entries is determined by the `finished_mutations_to_keep` storage engine parameter). Older mutation entries are deleted.

## Synchronicity of ALTER Queries

For non-replicated tables, all `ALTER` queries are performed synchronously. For replicated tables, the query just adds instructions for the appropriate actions to ZooKeeper, and the actions themselves are performed as soon as possible. However, the query can wait for these actions to be completed on all the replicas.

For all `ALTER` queries, you can use the `replication_alter_partitions_sync` setting to set up waiting.

You can specify how long (in seconds) to wait for inactive replicas to execute all `ALTER` queries with the `replication_wait_for_inactive_replica_timeout` setting.

### Note

For all `ALTER` queries, if `replication_alter_partitions_sync = 2` and some replicas are not active for more than the time, specified in the `replication_wait_for_inactive_replica_timeout` setting, then an exception `UNFINISHED` is thrown.

For `ALTER TABLE ... UPDATE|DELETE` queries the synchronicity is defined by the `mutations_sync` setting.

## Column Manipulations

A set of queries that allow changing the table structure.

Syntax:

```
ALTER TABLE [db].name [ON CLUSTER cluster] ADD|DROP|RENAME|CLEAR|COMMENT|MODIFY|MATERIALIZE COLUMN  
...
```

In the query, specify a list of one or more comma-separated actions.

Each action is an operation on a column.

The following actions are supported:

- **ADD COLUMN** — Adds a new column to the table.

- **DROP COLUMN** — Deletes the column.
- **RENAME COLUMN** — Renames an existing 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.
- **MODIFY COLUMN REMOVE** — Removes one of the column properties.
- **MATERIALIZE COLUMN** — Materializes the column in the parts where the column is missing.

These actions are described in detail below.

## ADD COLUMN

```
ADD COLUMN [IF NOT EXISTS] name [type] [default_expr] [codec] [AFTER name_after | FIRST]
```

Adds a new column to the table with the specified `name`, `type`, `codec` and `default_expr` (see the section [Default expressions](#)).

If the `IF NOT EXISTS` clause is included, the query won't return an error if the column already exists. If you specify `AFTER name_after` (the name of another column), the column is added after the specified one in the list of table columns. If you want to add a column to the beginning of the table use the `FIRST` clause. Otherwise, the column is added to the end of the table. For a chain of actions, `name_after` can be the name of a column that is added in one of the previous actions.

Adding a column just changes the table structure, without performing any actions with data. The data does not appear on the disk after `ALTER`. If the data is missing for a column when reading from the table, it is filled in with default values (by performing the default expression if there is one, or using zeros or empty strings). The column appears on the disk after merging data parts (see [MergeTree](#)).

This approach allows us to complete the `ALTER` query instantly, without increasing the volume of old data.

Example:

```
ALTER TABLE alter_test ADD COLUMN Added1 UInt32 FIRST;
ALTER TABLE alter_test ADD COLUMN Added2 UInt32 AFTER NestedColumn;
ALTER TABLE alter_test ADD COLUMN Added3 UInt32 AFTER ToDrop;
DESC alter_test FORMAT TSV;
```

```
Added1 UInt32
CounterID UInt32
StartDate Date
UserID UInt32
VisitID UInt32
NestedColumn.A Array(UInt8)
NestedColumn.S Array(String)
Added2 UInt32
ToDelete UInt32
Added3 UInt32
```

## DROP COLUMN

```
DROP COLUMN [IF EXISTS] name
```

Deletes the column with the name `name`. If the `IF EXISTS` clause is specified, the query won't return an error if the column does not exist.

Deletes data from the file system. Since this deletes entire files, the query is completed almost instantly.

## Warning

You can't delete a column if it is referenced by **materialized view**. Otherwise, it returns an error.

Example:

```
ALTER TABLE visits DROP COLUMN browser
```

## RENAME COLUMN

```
RENAME COLUMN [IF EXISTS] name to new_name
```

Renames the column `name` to `new_name`. If the `IF EXISTS` clause is specified, the query won't return an error if the column does not exist. Since renaming does not involve the underlying data, the query is completed almost instantly.

**NOTE:** Columns specified in the key expression of the table (either with `ORDER BY` or `PRIMARY KEY`) cannot be renamed. Trying to change these columns will produce [SQL Error \[524\]](#).

Example:

```
ALTER TABLE visits RENAME COLUMN webBrowser TO browser
```

## CLEAR COLUMN

```
CLEAR COLUMN [IF EXISTS] name IN PARTITION partition_name
```

Resets all data in a column for a specified partition. Read more about setting the partition name in the section [How to specify the partition expression](#).

If the `IF EXISTS` clause is specified, the query won't return an error if the column does not exist.

Example:

```
ALTER TABLE visits CLEAR COLUMN browser IN PARTITION tuple()
```

## COMMENT COLUMN

```
COMMENT COLUMN [IF EXISTS] name 'Text comment'
```

Adds a comment to the column. If the `IF EXISTS` clause is specified, the query won't return an error if the column does not exist.

Each column can have one comment. If a comment already exists for the column, a new comment overwrites the previous comment.

Comments are stored in the `comment_expression` column returned by the [DESCRIBE TABLE](#) query.

Example:

```
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] [codec] [TTL] [AFTER name_after | FIRST]
```

This query changes the `name` column properties:

- Type
- Default expression
- Compression Codec
- TTL

For examples of columns compression CODECS modifying, see [Column Compression Codecs](#).

For examples of columns TTL modifying, see [Column TTL](#).

If the `IF EXISTS` clause is specified, the query won't return an error if the column does not exist.

The query also can change the order of the columns using `FIRST | AFTER` clause, see [ADD COLUMN](#) description.

When changing the type, values are converted as if the `toType` functions were applied to them. If only the default expression is changed, the query does not do anything complex, and is completed almost instantly.

Example:

```
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.

The `ALTER` query is atomic. For MergeTree tables it is also lock-free.

The `ALTER` query for changing columns is replicated. The instructions are saved in ZooKeeper, then each replica applies them. All `ALTER` queries are run in the same order. The query waits for the appropriate actions to be completed on the other replicas. However, a query to change columns in a replicated table can be interrupted, and all actions will be performed asynchronously.

## MODIFY COLUMN REMOVE

Removes one of the column properties: `DEFAULT`, `ALIAS`, `MATERIALIZED`, `CODEC`, `COMMENT`, `TTL`.

Syntax:

```
ALTER TABLE table_name MODIFY column_name REMOVE property;
```

### Example

Remove TTL:

```
ALTER TABLE table_with_ttl MODIFY COLUMN column_ttl REMOVE TTL;
```

## See Also

- [REMOVE TTL](#).

# MATERIALIZE COLUMN

Materializes the column in the parts where the column is missing. This is useful in case of creating a new column with complicated `DEFAULT` or `MATERIALIZED` expression. Calculation of the column directly on `SELECT` query can cause bigger request execution time, so it is reasonable to use `MATERIALIZE COLUMN` for such columns. To perform same manipulation for existing column, use `FINAL` modifier.

Syntax:

```
ALTER TABLE table MATERIALIZE COLUMN col [FINAL];
```

## Example

```
DROP TABLE IF EXISTS tmp;
SET mutations_sync = 2;
CREATE TABLE tmp (x Int64) ENGINE = MergeTree() ORDER BY tuple() PARTITION BY tuple();
INSERT INTO tmp SELECT * FROM system.numbers LIMIT 10;
ALTER TABLE tmp ADD COLUMN s String MATERIALIZED toString(x);
SELECT groupArray(x), groupArray(s) FROM tmp;
```

## Result:

groupArray(x)	groupArray(s)
[0,1,2,3,4,5,6,7,8,9]	['0','1','2','3','4','5','6','7','8','9']

## See Also

- [MATERIALIZED](#).

# Limitations

The `ALTER` query lets you create and delete separate elements (columns) in nested data structures, but not whole nested data structures. To add a nested data structure, you can add columns with a name like `name.nested_name` and the type `Array(T)`. A nested data structure is equivalent to multiple array columns with a name that has the same prefix before the dot.

There is no support for deleting columns in the primary key or the sampling key (columns that are used in the `ENGINE` expression). Changing the type for columns that are included in the primary key is only possible if this change does not cause the data to be modified (for example, you are allowed to add values to an `Enum` or to change a type from `DateTime` to `UInt32`).

If the `ALTER` query is not sufficient to make the table changes you need, you can create a new table, copy the data to it using the `INSERT SELECT` query, then switch the tables using the `RENAME` query and delete the old table. You can use the [clickhouse-copier](#) as an alternative to the `INSERT SELECT` query.

The `ALTER` query blocks all reads and writes for the table. In other words, if a long `SELECT` is running at the time of the `ALTER` query, the `ALTER` query will wait for it to complete. At the same time, all new queries to the same table will wait while this `ALTER` is running.

For tables that do not store data themselves (such as [Merge](#) and [Distributed](#)), [ALTER](#) just changes the table structure, and does not change the structure of subordinate tables. For example, when running [ALTER](#) for a [Distributed](#) table, you will also need to run [ALTER](#) for the tables on all remote servers.

## Manipulating Partitions and Parts

The following operations with [partitions](#) are available:

- [DETACH PARTITION](#) — Moves a partition to the [detached](#) directory and forget it.
- [DROP PARTITION](#) — Deletes a partition.
- [ATTACH PART|PARTITION](#) — Adds a part or partition from the [detached](#) directory to the table.
- [ATTACH PARTITION FROM](#) — Copies the data partition from one table to another and adds.
- [REPLACE PARTITION](#) — Copies the data partition from one table to another and replaces.
- [MOVE PARTITION TO TABLE](#) — Moves the data partition from one table to another.
- [CLEAR COLUMN IN PARTITION](#) — Resets the value of a specified column in a partition.
- [CLEAR INDEX IN PARTITION](#) — Resets the specified secondary index in a partition.
- [FREEZE PARTITION](#) — Creates a backup of a partition.
- [UNFREEZE PARTITION](#) — Removes a backup of a partition.
- [FETCH PARTITION|PART](#) — Downloads a part or partition from another server.
- [MOVE PARTITION|PART](#) — Move partition/data part to another disk or volume.
- [UPDATE IN PARTITION](#) — Update data inside the partition by condition.
- [DELETE IN PARTITION](#) — Delete data inside the partition by condition.

### DETACH PARTITION|PART

```
ALTER TABLE table_name DETACH PARTITION|PART partition_expr
```

Moves all data for the specified partition to the [detached](#) directory. The server forgets about the detached data partition as if it does not exist. The server will not know about this data until you make the [ATTACH](#) query.

Example:

```
ALTER TABLE mt DETACH PARTITION '2020-11-21';
ALTER TABLE mt DETACH PART 'all_2_2_0';
```

Read about setting the partition expression in a section [How to specify the partition expression](#).

After the query is executed, you can do whatever you want with the data in the [detached](#) directory — delete it from the file system, or just leave it.

This query is replicated — it moves the data to the [detached](#) directory on all replicas. Note that you can execute this query only on a leader replica. To find out if a replica is a leader, perform the [SELECT](#) query to the [system.replicas](#) table. Alternatively, it is easier to make a [DETACH](#) query on all replicas - all the replicas throw an exception, except the leader replicas (as multiple leaders are allowed).

## DROP PARTITION|PART

```
ALTER TABLE table_name DROP PARTITION|PART partition_expr
```

Deletes the specified partition from the table. This query tags the partition as inactive and deletes data completely, approximately in 10 minutes.

Read about setting the partition expression in a section [How to specify the partition expression](#).

The query is replicated – it deletes data on all replicas.

Example:

```
ALTER TABLE mt DROP PARTITION '2020-11-21';
ALTER TABLE mt DROP PART 'all_4_4_0';
```

## DROP DETACHED PARTITION|PART

```
ALTER TABLE table_name DROP DETACHED PARTITION|PART partition_expr
```

Removes the specified part or all parts of the specified partition from `detached`.

Read more about setting the partition expression in a section [How to specify the partition expression](#).

## ATTACH PARTITION|PART

```
ALTER TABLE table_name ATTACH PARTITION|PART partition_expr
```

Adds data to the table from the `detached` directory. It is possible to add data for an entire partition or for a separate part. Examples:

```
ALTER TABLE visits ATTACH PARTITION 201901;
ALTER TABLE visits ATTACH PART 201901_2_2_0;
```

Read more about setting the partition expression in a section [How to specify the partition expression](#).

This query is replicated. The replica-initiator checks whether there is data in the `detached` directory.

If data exists, the query checks its integrity. If everything is correct, the query adds the data to the table.

If the non-initiator replica, receiving the attach command, finds the part with the correct checksums in its own `detached` folder, it attaches the data without fetching it from other replicas.

If there is no part with the correct checksums, the data is downloaded from any replica having the part.

You can put data to the `detached` directory on one replica and use the `ALTER ... ATTACH` query to add it to the table on all replicas.

## ATTACH PARTITION FROM

```
ALTER TABLE table2 ATTACH PARTITION partition_expr FROM table1
```

This query copies the data partition from `table1` to `table2`.

Note that data will be deleted neither from `table1` nor from `table2`.

For the query to run successfully, the following conditions must be met:

- Both tables must have the same structure.
- Both tables must have the same partition key.

## REPLACE PARTITION

```
ALTER TABLE table2 REPLACE PARTITION partition_expr FROM table1
```

This query copies the data partition from the `table1` to `table2` and replaces existing partition in the `table2`. Note that data won't be deleted from `table1`.

For the query to run successfully, the following conditions must be met:

- Both tables must have the same structure.
- Both tables must have the same partition key.

## MOVE PARTITION TO TABLE

```
ALTER TABLE table_source MOVE PARTITION partition_expr TO TABLE table_dest
```

This query moves the data partition from the `table_source` to `table_dest` with deleting the data from `table_source`.

For the query to run successfully, the following conditions must be met:

- Both tables must have the same structure.
- Both tables must have the same partition key.
- Both tables must be the same engine family (replicated or non-replicated).
- Both tables must have the same storage policy.

## CLEAR COLUMN IN PARTITION

```
ALTER TABLE table_name CLEAR COLUMN column_name IN PARTITION partition_expr
```

Resets all values in the specified column in a partition. If the `DEFAULT` clause was determined when creating a table, this query sets the column value to a specified default value.

Example:

```
ALTER TABLE visits CLEAR COLUMN hour IN PARTITION 201902
```

## FREEZE PARTITION

```
ALTER TABLE table_name FREEZE [PARTITION partition_expr]
```

This query creates a local backup of a specified partition. If the `PARTITION` clause is omitted, the query creates the backup of all partitions at once.

### Note

The entire backup process is performed without stopping the server.

Note that for old-styled tables you can specify the prefix of the partition name (for example, `2019`) - then the query creates the backup for all the corresponding partitions. Read about setting the partition expression in a section [How to specify the partition expression](#).

At the time of execution, for a data snapshot, the query creates hardlinks to a table data. Hardlinks are placed in the directory `/var/lib/clickhouse/shadow/N/...`, where:

- `/var/lib/clickhouse/` is the working ClickHouse directory specified in the config.
- `N` is the incremental number of the backup.

## Note

If you use **a set of disks for data storage in a table**, the `shadow/N` directory appears on every disk, storing data parts that matched by the `PARTITION` expression.

The same structure of directories is created inside the backup as inside `/var/lib/clickhouse/`. The query performs `chmod` for all files, forbidding writing into them.

After creating the backup, you can copy the data from `/var/lib/clickhouse/shadow/` to the remote server and then delete it from the local server. Note that the `ALTER t FREEZE PARTITION` query is not replicated. It creates a local backup only on the local server.

The query creates backup almost instantly (but first it waits for the current queries to the corresponding table to finish running).

`ALTER TABLE t FREEZE PARTITION` copies only the data, not table metadata. To make a backup of table metadata, copy the file `/var/lib/clickhouse/metadata/database/table.sql`

To restore data from a backup, do the following:

1. Create the table if it does not exist. To view the query, use the `.sql` file (replace `ATTACH` in it with `CREATE`).
2. Copy the data from the `data/database/table/` directory inside the backup to the `/var/lib/clickhouse/data/database/table/detached/` directory.
3. Run `ALTER TABLE t ATTACH PARTITION` queries to add the data to a table.

Restoring from a backup does not require stopping the server.

For more information about backups and restoring data, see the [Data Backup](#) section.

## UNFREEZE PARTITION

```
ALTER TABLE 'table_name' UNFREEZE [PARTITION 'part_expr'] WITH NAME 'backup_name'
```

Removes freezed partitions with the specified name from the disk. If the `PARTITION` clause is omitted, the query removes the backup of all partitions at once.

## CLEAR INDEX IN PARTITION

```
ALTER TABLE table_name CLEAR INDEX index_name IN PARTITION partition_expr
```

The query works similar to `CLEAR COLUMN`, but it resets an index instead of a column data.

# FETCH PARTITION|PART

```
ALTER TABLE table_name FETCH PARTITION|PART partition_expr FROM 'path-in-zookeeper'
```

Downloads a partition from another server. This query only works for the replicated tables.

The query does the following:

1. Downloads the partition|part from the specified shard. In 'path-in-zookeeper' you must specify a path to the shard in ZooKeeper.
2. Then the query puts the downloaded data to the `detached` directory of the `table_name` table. Use the **ATTACH PARTITION|PART** query to add the data to the table.

For example:

1. **FETCH PARTITION**

```
ALTER TABLE users FETCH PARTITION 201902 FROM '/clickhouse/tables/01-01/visits';
ALTER TABLE users ATTACH PARTITION 201902;
```

2. **FETCH PART**

```
ALTER TABLE users FETCH PART 201901_2_2_0 FROM '/clickhouse/tables/01-01/visits';
ALTER TABLE users ATTACH PART 201901_2_2_0;
```

Note that:

- The `ALTER ... FETCH PARTITION|PART` query isn't replicated. It places the part or partition to the `detached` directory only on the local server.
- The `ALTER TABLE ... ATTACH` query is replicated. It adds the data to all replicas. The data is added to one of the replicas from the `detached` directory, and to the others - from neighboring replicas.

Before downloading, the system checks if the partition exists and the table structure matches. The most appropriate replica is selected automatically from the healthy replicas.

Although the query is called `ALTER TABLE`, it does not change the table structure and does not immediately change the data available in the table.

# MOVE PARTITION|PART

Moves partitions or data parts to another volume or disk for `MergeTree`-engine tables. See [Using Multiple Block Devices for Data Storage](#).

```
ALTER TABLE table_name MOVE PARTITION|PART partition_expr TO DISK|VOLUME 'disk_name'
```

The `ALTER TABLE t MOVE` query:

- Not replicated, because different replicas can have different storage policies.
- Returns an error if the specified disk or volume is not configured. Query also returns an error if conditions of data moving, that specified in the storage policy, can't be applied.
- Can return an error in the case, when data to be moved is already moved by a background process, concurrent `ALTER TABLE t MOVE` query or as a result of background data merging. A user shouldn't perform any additional actions in this case.

Example:

```
ALTER TABLE hits MOVE PART '20190301_14343_16206_438' TO VOLUME 'slow'  
ALTER TABLE hits MOVE PARTITION '2019-09-01' TO DISK 'fast_ssd'
```

## UPDATE IN PARTITION

Manipulates data in the specifies partition matching the specified filtering expression. Implemented as a [mutation](#).

Syntax:

```
ALTER TABLE [db.]table UPDATE column1 = expr1 [, ...] [IN PARTITION partition_id] WHERE filter_expr
```

### Example

```
ALTER TABLE mt UPDATE x = x + 1 IN PARTITION 2 WHERE p = 2;
```

### See Also

- [UPDATE](#)

## DELETE IN PARTITION

Deletes data in the specifies partition matching the specified filtering expression. Implemented as a [mutation](#).

Syntax:

```
ALTER TABLE [db.]table DELETE [IN PARTITION partition_id] WHERE filter_expr
```

### Example

```
ALTER TABLE mt DELETE IN PARTITION 2 WHERE p = 2;
```

### See Also

- [DELETE](#)

## How to Set Partition Expression

You can specify the partition expression in `ALTER ... PARTITION` queries in different ways:

- As a value from the `partition` column of the `system.parts` table. For example, `ALTER TABLE visits DETACH PARTITION 201901`.
- As a tuple of expressions or constants that matches (in types) the table partitioning keys tuple. In the case of a single element partitioning key, the expression should be wrapped in the `tuple(...)` function. For example, `ALTER TABLE visits DETACH PARTITION tuple(toYYYYMM(toDate('2019-01-25')))`.
- Using the partition ID. Partition ID is a string identifier of the partition (human-readable, if possible) that is used as the names of partitions in the file system and in ZooKeeper. The partition ID must be specified in the `PARTITION ID` clause, in a single quotes. For example, `ALTER TABLE visits DETACH PARTITION ID '201901'`.

- In the **ALTER ATTACH PART** and **DROP DETACHED PART** query, to specify the name of a part, use string literal with a value from the `name` column of the `system.detached_parts` table. For example, `ALTER TABLE visits ATTACH PART '201901_1_1_0'`.

Usage of quotes when specifying the partition depends on the type of partition expression. For example, for the `String` type, you have to specify its name in quotes (''). For the `Date` and `Int*` types no quotes are needed.

All the rules above are also true for the **OPTIMIZE** query. If you need to specify the only partition when optimizing a non-partitioned table, set the expression `PARTITION tuple()`. For example:

```
OPTIMIZE TABLE table_not_partitioned PARTITION tuple() FINAL;
```

`IN PARTITION` specifies the partition to which the **UPDATE** or **DELETE** expressions are applied as a result of the `ALTER TABLE` query. New parts are created only from the specified partition. In this way, `IN PARTITION` helps to reduce the load when the table is divided into many partitions, and you only need to update the data point-by-point.

The examples of `ALTER ... PARTITION` queries are demonstrated in the tests `00502_custom_partitioning_local` and `00502_custom_partitioning_replicated_zookeeper`.

## Table Settings Manipulations

There is a set of queries to change table settings. You can modify settings or reset them to default values. A single query can change several settings at once.

If a setting with the specified name does not exist, then the query raises an exception.

### Syntax

```
ALTER TABLE [db].name [ON CLUSTER cluster] MODIFY|RESET SETTING ...
```

### Note

These queries can be applied to **MergeTree** tables only.

## MODIFY SETTING

Changes table settings.

### Syntax

```
MODIFY SETTING setting_name=value [, ...]
```

### Example

```
CREATE TABLE example_table (id UInt32, data String) ENGINE=MergeTree() ORDER BY id;  
ALTER TABLE example_table MODIFY SETTING max_part_loading_threads=8, max_parts_in_total=50000;
```

## RESET SETTING

Resets table settings to their default values. If a setting is in a default state, then no action is taken.

## Syntax

```
RESET SETTING setting_name [, ...]
```

## Example

```
CREATE TABLE example_table (id UInt32, data String) ENGINE=MergeTree() ORDER BY id  
SETTINGS max_part_loading_threads=8;
```

```
ALTER TABLE example_table RESET SETTING max_part_loading_threads;
```

## See Also

- [MergeTree settings](#)

# ALTER TABLE ... DELETE Statement

```
ALTER TABLE [db.]table [ON CLUSTER cluster] DELETE WHERE filter_expr
```

Deletes data matching the specified filtering expression. Implemented as a [mutation](#).

## Note

The `ALTER TABLE` prefix makes this syntax different from most other systems supporting SQL. It is intended to signify that unlike similar queries in OLTP databases this is a heavy operation not designed for frequent use.

The `filter_expr` must be of type `UInt8`. The query deletes rows in the table for which this expression takes a non-zero value.

One query can contain several commands separated by commas.

The synchronicity of the query processing is defined by the `mutations_sync` setting. By default, it is asynchronous.

## See also

- [Mutations](#)
- [Synchronicity of ALTER Queries](#)
- [mutations\\_sync setting](#)

# ALTER TABLE ... UPDATE Statements

```
ALTER TABLE [db.]table UPDATE column1 = expr1 [, ...] WHERE filter_expr
```

Manipulates data matching the specified filtering expression. Implemented as a [mutation](#).

## Note

The `ALTER TABLE` prefix makes this syntax different from most other systems supporting SQL. It is intended to signify that unlike similar queries in OLTP databases this is a heavy operation not designed for frequent use.

The `filter_expr` must be of type `UInt8`. This query updates values of specified columns to the values of corresponding expressions in rows for which the `filter_expr` takes a non-zero value. Values are casted to the column type using the `CAST` operator. Updating columns that are used in the calculation of the primary or the partition key is not supported.

One query can contain several commands separated by commas.

The synchronicity of the query processing is defined by the `mutations_sync` setting. By default, it is asynchronous.

## See also

- [Mutations](#)
- [Synchronicity of ALTER Queries](#)
- [mutations\\_sync setting](#)

# Manipulating Key Expressions

```
ALTER TABLE [db].name [ON CLUSTER cluster] MODIFY ORDER BY new_expression
```

The command changes the `sorting key` of the table to `new_expression` (an expression or a tuple of expressions). Primary key remains the same.

The command is lightweight in a sense that it only changes metadata. To keep the property that data part rows are ordered by the sorting key expression you cannot add expressions containing existing columns to the sorting key (only columns added by the `ADD COLUMN` command in the same `ALTER` query, without default column value).

## Note

It only works for tables in the `MergeTree` family (including `replicated` tables).

# Manipulating Sampling-Key Expressions

Syntax:

```
ALTER TABLE [db].name [ON CLUSTER cluster] MODIFY SAMPLE BY new_expression
```

The command changes the `sampling key` of the table to `new_expression` (an expression or a tuple of expressions).

The command is lightweight in the sense that it only changes metadata. The primary key must contain the new sample key.

## Note

It only works for tables in the **MergeTree** family (including **replicated** tables).

# Manipulating Data Skipping Indices

The following operations are available:

- `ALTER TABLE [db].name ADD INDEX name expression TYPE type GRANULARITY value [FIRST|AFTER name]` - Adds index description to tables metadata.
- `ALTER TABLE [db].name DROP INDEX name` - Removes index description from tables metadata and deletes index files from disk.
- `ALTER TABLE [db.]table MATERIALIZE INDEX name IN PARTITION partition_name` - The query rebuilds the secondary index `name` in the partition `partition_name`. Implemented as a **mutation**. To rebuild index over the whole data in the table you need to remove `IN PARTITION` from query.

The first two commands are lightweight in a sense that they only change metadata or remove files.

Also, they are replicated, syncing indices metadata via ZooKeeper.

## Note

Index manipulation is supported only for tables with **\*MergeTree** engine (including **replicated** variants).

# Manipulating Constraints

Constraints could be added or deleted using following syntax:

```
ALTER TABLE [db].name ADD CONSTRAINT constraint_name CHECK expression;
ALTER TABLE [db].name DROP CONSTRAINT constraint_name;
```

See more on [constraints](#).

Queries will add or remove metadata about constraints from table so they are processed immediately.

## Warning

Constraint check **will not be executed** on existing data if it was added.

All changes on replicated tables are broadcasted to ZooKeeper and will be applied on other replicas as well.

# Manipulations with Table TTL

## MODIFY TTL

You can change `table TTL` with a request of the following form:

```
ALTER TABLE table_name MODIFY TTL ttl_expression;
```

## REMOVE TTL

TTL-property can be removed from table with the following query:

```
ALTER TABLE table_name REMOVE TTL
```

### Example

Consider the table with table TTL:

```
CREATE TABLE table_with_ttl
(
    event_time DateTime,
    UserID UInt64,
    Comment String
)
ENGINE MergeTree()
ORDER BY tuple()
TTL event_time + INTERVAL 3 MONTH;
SETTINGS min_bytes_for_wide_part = 0;

INSERT INTO table_with_ttl VALUES (now(), 1, 'username1');

INSERT INTO table_with_ttl VALUES (now() - INTERVAL 4 MONTH, 2, 'username2');
```

Run OPTIMIZE to force TTL cleanup:

```
OPTIMIZE TABLE table_with_ttl FINAL;
SELECT * FROM table_with_ttl FORMAT PrettyCompact;
```

Second row was deleted from table.

event_time	UserID	Comment
2020-12-11 12:44:57	1	username1

Now remove table TTL with the following query:

```
ALTER TABLE table_with_ttl REMOVE TTL;
```

Re-insert the deleted row and force the TTL cleanup again with OPTIMIZE:

```
INSERT INTO table_with_ttl VALUES (now() - INTERVAL 4 MONTH, 2, 'username2');
OPTIMIZE TABLE table_with_ttl FINAL;
SELECT * FROM table_with_ttl FORMAT PrettyCompact;
```

The TTL is no longer there, so the second row is not deleted:

event_time	UserID	Comment
2020-12-11 12:44:57	1	username1
2020-08-11 12:44:57	2	username2

### See Also

- More about the [TTL-expression](#).
- Modify column [with TTL](#).

# ALTER USER

Changes ClickHouse user accounts.

Syntax:

```
ALTER USER [IF EXISTS] name1 [ON CLUSTER cluster_name1] [RENAME TO new_name1]
    [, name2 [ON CLUSTER cluster_name2] [RENAME TO new_name2] ...]
    [NOT IDENTIFIED | IDENTIFIED {[WITH {no_password | plaintext_password | sha256_password | sha256_hash |
    double_sha1_password | double_sha1_hash}] BY {'password' | 'hash'}} | {WITH ldap SERVER 'server_name'} | {WITH
    kerberos [REALM 'realm']}]
    {[ADD | DROP] HOST {LOCAL | NAME 'name' | REGEXP 'name_regex' | IP 'address' | LIKE 'pattern'} [...], ANY | 
    NONE}
    [DEFAULT ROLE role [...] | ALL | ALL EXCEPT role [...] ]
    [GRANTEES {user | role | ANY | NONE} [...] [EXCEPT {user | role} [...]]]
    [SETTINGS variable [= value] [MIN [=] min_value] [MAX [=] max_value] [READONLY | WRITABLE] | PROFILE
    'profile_name'] [...]
```

To use `ALTER USER` you must have the [ALTER USER](#) privilege.

## GRANTEES Clause

Specifies users or roles which are allowed to receive [privileges](#) from this user on the condition this user has also all required access granted with [GRANT OPTION](#). Options of the GRANTEES clause:

- `user` — Specifies a user this user can grant privileges to.
- `role` — Specifies a role this user can grant privileges to.
- `ANY` — This user can grant privileges to anyone. It's the default setting.
- `NONE` — This user can grant privileges to none.

You can exclude any user or role by using the `EXCEPT` expression. For example, `ALTER USER user1 GRANTEES ANY EXCEPT user2`. It means if `user1` has some privileges granted with `GRANT OPTION` it will be able to grant those privileges to anyone except `user2`.

## Examples

Set assigned roles as default:

```
ALTER USER user DEFAULT ROLE role1, role2
```

If roles aren't previously assigned to a user, ClickHouse throws an exception.

Set all the assigned roles to default:

```
ALTER USER user DEFAULT ROLE ALL
```

If a role is assigned to a user in the future, it will become default automatically.

Set all the assigned roles to default, excepting `role1` and `role2`:

```
ALTER USER user DEFAULT ROLE ALL EXCEPT role1, role2
```

Allows the user with `john` account to grant his privileges to the user with `jack` account:

```
ALTER USER john GRANTEES jack;
```

## ALTER QUOTA

Changes quotas.

Syntax:

```
ALTER QUOTA [IF EXISTS] name [ON CLUSTER cluster_name]
[RENAME TO new_name]
[KEYED BY {user_name | ip_address | client_key | client_key,user_name | client_key,ip_address} | NOT KEYED]
[FOR [RANDOMIZED] INTERVAL number {second | minute | hour | day | week | month | quarter | year}
 {MAX { {queries | query_selects | query_inserts | errors | result_rows | result_bytes | read_rows | read_bytes | execution_time} = number } [...] | NO LIMITS | TRACKING ONLY} [...]]
[TO {role [...] | ALL | ALL EXCEPT role [...]}]
```

Keys `user_name`, `ip_address`, `client_key`, `client_key, user_name` and `client_key, ip_address` correspond to the fields in the [system.quotas](#) table.

Parameters `queries`, `query_selects`, '`query_inserts`', `errors`,`result_rows`,`result_bytes`,`read_rows`,`read_bytes`,`execution_time`` correspond to the fields in the [system.quotas\\_usage](#) table.

`ON CLUSTER` clause allows creating quotas on a cluster, see [Distributed DDL](#).

### Examples

Limit the maximum number of queries for the current user with 123 queries in 15 months constraint:

```
ALTER QUOTA IF EXISTS qA FOR INTERVAL 15 month MAX queries = 123 TO CURRENT_USER;
```

For the default user limit the maximum execution time with half a second in 30 minutes, and limit the maximum number of queries with 321 and the maximum number of errors with 10 in 5 quarters:

```
ALTER QUOTA IF EXISTS qB FOR INTERVAL 30 minute MAX execution_time = 0.5, FOR INTERVAL 5 quarter MAX queries = 321, errors = 10 TO default;
```

## ALTER ROLE

Changes roles.

Syntax:

```
ALTER ROLE [IF EXISTS] name1 [ON CLUSTER cluster_name1] [RENAME TO new_name1]
[, name2 [ON CLUSTER cluster_name2] [RENAME TO new_name2] ...]
[SETTINGS variable [= value] [MIN [=] min_value] [MAX [=] max_value] [READONLY|WRITABLE] | PROFILE
'profile_name'] [...]
```

## ALTER ROW POLICY

Changes row policy.

Syntax:

```
ALTER [ROW] POLICY [IF EXISTS] name1 [ON CLUSTER cluster_name1] ON [database1.]table1 [RENAME TO new_name1]
    [, name2 [ON CLUSTER cluster_name2] ON [database2.]table2 [RENAME TO new_name2] ...]
    [AS {PERMISSIVE | RESTRICTIVE}]
    [FOR SELECT]
    [USING {condition | NONE}] [...]
    [TO {role [...] | ALL | ALL EXCEPT role [...]}]
```

## ALTER SETTINGS PROFILE

Changes settings profiles.

Syntax:

```
ALTER SETTINGS PROFILE [IF EXISTS] TO name1 [ON CLUSTER cluster_name1] [RENAME TO new_name1]
    [, name2 [ON CLUSTER cluster_name2] [RENAME TO new_name2] ...]
    [SETTINGS variable [= value] [MIN [=] min_value] [MAX [=] max_value] [READONLY|WRITABLE] | INHERIT 'profile_name'] [...]
```

## Manipulating Projections

The following operations with **projections** are available:

- `ALTER TABLE [db].name ADD PROJECTION name ( SELECT <COLUMN LIST EXPR> [GROUP BY] [ORDER BY] )` - Adds projection description to tables metadata.
- `ALTER TABLE [db].name DROP PROJECTION name` - Removes projection description from tables metadata and deletes projection files from disk.
- `ALTER TABLE [db.]table MATERIALIZE PROJECTION name IN PARTITION partition_name` - The query rebuilds the projection name in the partition `partition_name`. Implemented as a **mutation**.
- `ALTER TABLE [db.]table CLEAR PROJECTION name IN PARTITION partition_name` - Deletes projection files from disk without removing description.

The commands `ADD`, `DROP` and `CLEAR` are lightweight in a sense that they only change metadata or remove files.

Also, they are replicated, syncing projections metadata via ZooKeeper.

### Note

Projection manipulation is supported only for tables with `*MergeTree` engine (including `replicated` variants).

## ALTER TABLE ... MODIFY QUERY Statement

You can modify `SELECT` query that was specified when a `materialized view` was created with the `ALTER TABLE ... MODIFY QUERY` statement. Use it when the materialized view was created without the `TO [db.]name` clause. The `allow_experimental_alter_materialized_view_structure` setting must be enabled.

If a materialized view uses the TO [db.]name construction, you must **DETACH** the view, run **ALTER TABLE** query for the target table, and then **ATTACH** the previously detached (**DETACH**) view.

## Example

```
CREATE TABLE src_table (`a` UInt32) ENGINE = MergeTree ORDER BY a;
CREATE MATERIALIZED VIEW mv (`a` UInt32) ENGINE = MergeTree ORDER BY a AS SELECT a FROM src_table;
INSERT INTO src_table (a) VALUES (1), (2);
SELECT * FROM mv;
```

a
1
2

```
ALTER TABLE mv MODIFY QUERY SELECT a * 2 as a FROM src_table;
INSERT INTO src_table (a) VALUES (3), (4);
SELECT * FROM mv;
```

a
6
8

a
1
2

## ALTER LIVE VIEW Statement

`ALTER LIVE VIEW ... REFRESH` statement refreshes a [Live view](#). See [Force Live View Refresh](#).

## SYSTEM Statements

The list of available `SYSTEM` statements:

- [RELOAD EMBEDDED DICTIONARIES](#)
- [RELOAD DICTIONARIES](#)
- [RELOAD DICTIONARY](#)
- [RELOAD MODELS](#)
- [RELOAD MODEL](#)
- [RELOAD FUNCTIONS](#)
- [RELOAD FUNCTION](#)
- [DROP DNS CACHE](#)
- [DROP MARK CACHE](#)
- [DROP UNCOMPRESSED CACHE](#)
- [DROP COMPILED EXPRESSION CACHE](#)
- [DROP REPLICA](#)

- FLUSH LOGS
- RELOAD CONFIG
- SHUTDOWN
- KILL
- STOP DISTRIBUTED SENDS
- FLUSH DISTRIBUTED
- START DISTRIBUTED SENDS
- STOP MERGES
- START MERGES
- STOP TTL MERGES
- START TTL MERGES
- STOP MOVES
- START MOVES
- STOP FETCHES
- START FETCHES
- STOP REPLICATED SENDS
- START REPLICATED SENDS
- STOP REPLICATION QUEUES
- START REPLICATION QUEUES
- SYNC REPLICA
- RESTART REPLICA
- RESTORE REPLICA
- RESTART REPLICAS

## RELOAD EMBEDDED DICTIONARIES

Reload all [Internal dictionaries](#).

By default, internal dictionaries are disabled.

Always returns `Ok`, regardless of the result of the internal dictionary update.

## RELOAD DICTIONARIES

Reloads all dictionaries that have been successfully loaded before.

By default, dictionaries are loaded lazily (see [dictionaries\\_lazy\\_load](#)), so instead of being loaded automatically at startup, they are initialized on first access through dictGet function or SELECT from tables with ENGINE = Dictionary. The `SYSTEM RELOAD DICTIONARIES` query reloads such dictionaries (LOADED). Always returns `Ok`, regardless of the result of the dictionary update.

## RELOAD DICTIONARY

Completely reloads a dictionary `dictionary_name`, regardless of the state of the dictionary (LOADED / NOT\_LOADED / FAILED).

Always returns `Ok`, regardless of the result of updating the dictionary.

The status of the dictionary can be checked by querying the `system.dictionaries` table.

```
SELECT name, status FROM system.dictionaries;
```

## RELOAD MODELS

Reloads all `CatBoost` models if the configuration was updated without restarting the server.

### Syntax

```
SYSTEM RELOAD MODELS
```

## RELOAD MODEL

Completely reloads a `CatBoost` model `model_name` if the configuration was updated without restarting the server.

### Syntax

```
SYSTEM RELOAD MODEL <model_name>
```

## RELOAD FUNCTIONS

Reloads all registered `executable user defined functions` or one of them from a configuration file.

### Syntax

```
RELOAD FUNCTIONS  
RELOAD FUNCTION function_name
```

## DROP DNS CACHE

Resets ClickHouse's internal DNS cache. Sometimes (for old ClickHouse versions) it is necessary to use this command when changing the infrastructure (changing the IP address of another ClickHouse server or the server used by dictionaries).

For more convenient (automatic) cache management, see `disable_internal_dns_cache`, `dns_cache_update_period` parameters.

## DROP MARK CACHE

Resets the mark cache. Used in development of ClickHouse and performance tests.

## DROP REPLICA

Dead replicas can be dropped using following syntax:

```
SYSTEM DROP REPLICA 'replica_name' FROM TABLE database.table;  
SYSTEM DROP REPLICA 'replica_name' FROM DATABASE database;  
SYSTEM DROP REPLICA 'replica_name';  
SYSTEM DROP REPLICA 'replica_name' FROM ZKPATH '/path/to/table/in/zk';
```

Queries will remove the replica path in ZooKeeper. It is useful when the replica is dead and its metadata cannot be removed from ZooKeeper by `DROP TABLE` because there is no such table anymore. It will only drop the inactive/stale replica, and it cannot drop local replica, please use `DROP TABLE` for that. `DROP REPLICA` does not drop any tables and does not remove any data or metadata from disk.

The first one removes metadata of '`replica_name`' replica of `database.table` table.

The second one does the same for all replicated tables in the database.

The third one does the same for all replicated tables on the local server.

The fourth one is useful to remove metadata of dead replica when all other replicas of a table were dropped. It requires the table path to be specified explicitly. It must be the same path as was passed to the first argument of `ReplicatedMergeTree` engine on table creation.

## DROP UNCOMPRESSED CACHE

Reset the uncompressed data cache. Used in development of ClickHouse and performance tests.

For manage uncompressed data cache parameters use following server level settings

`uncompressed_cache_size` and query/user/profile level settings `use_uncompressed_cache`

## DROP COMPILED EXPRESSION CACHE

Reset the compiled expression cache. Used in development of ClickHouse and performance tests.

Compiled expression cache used when query/user/profile enable option `compile-expressions`

## 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.

This will also create system tables even if message queue is empty.

## RELOAD CONFIG

Reloads ClickHouse configuration. Used when configuration is stored in ZooKeeper.

## SHUTDOWN

Normally shuts down ClickHouse (like `service clickhouse-server stop / kill {$pid_clickhouse-server}`)

## KILL

Aborts ClickHouse process (like `kill -9 {$pid_clickhouse-server}`)

## Managing Distributed Tables

ClickHouse can manage **distributed** tables. When a user inserts data into these tables, ClickHouse first creates a queue of the data that should be sent to cluster nodes, then asynchronously sends it. You can manage queue processing with the `STOP DISTRIBUTED SENDS`, `FLUSH DISTRIBUTED`, and `START DISTRIBUTED SENDS` queries. You can also synchronously insert distributed data with the `insert_distributed_sync` setting.

## STOP DISTRIBUTED SENDS

Disables background data distribution when inserting data into distributed tables.

```
SYSTEM STOP DISTRIBUTED SENDS [db.]<distributed_table_name>
```

## FLUSH DISTRIBUTED

Forces ClickHouse to send data to cluster nodes synchronously. If any nodes are unavailable, ClickHouse throws an exception and stops query execution. You can retry the query until it succeeds, which will happen when all nodes are back online.

```
SYSTEM FLUSH DISTRIBUTED [db.]<distributed_table_name>
```

## START DISTRIBUTED SENDS

Enables background data distribution when inserting data into distributed tables.

```
SYSTEM START DISTRIBUTED SENDS [db.]<distributed_table_name>
```

## Managing MergeTree Tables

ClickHouse can manage background processes in [MergeTree](#) tables.

### STOP MERGES

Provides possibility to stop background merges for tables in the MergeTree family:

```
SYSTEM STOP MERGES [ON VOLUME <volume_name> | [db.]merge_tree_family_table_name]
```

#### Note

`DETACH / ATTACH` table will start background merges for the table even in case when merges have been stopped for all MergeTree tables before.

### START MERGES

Provides possibility to start background merges for tables in the MergeTree family:

```
SYSTEM START MERGES [ON VOLUME <volume_name> | [db.]merge_tree_family_table_name]
```

### STOP TTL MERGES

Provides possibility to stop background delete old data according to [TTL expression](#) for tables in the MergeTree family:

Returns `Ok.` even if table does not exist or table has not MergeTree engine. Returns error when database does not exist:

```
SYSTEM STOP TTL MERGES [[db.]merge_tree_family_table_name]
```

### START TTL MERGES

Provides possibility to start background delete old data according to [TTL expression](#) for tables in the MergeTree family:

Returns `Ok.` even if table does not exist. Returns error when database does not exist:

```
SYSTEM START TTL MERGES [[db.]merge_tree_family_table_name]
```

### STOP MOVES

Provides possibility to stop background move data according to [TTL table expression with TO VOLUME or TO DISK clause](#) for tables in the MergeTree family:

Returns Ok. even if table does not exist. Returns error when database does not exist:

```
SYSTEM STOP MOVES [[db.]merge_tree_family_table_name]
```

## START MOVES

Provides possibility to start background move data according to [TTL table expression with TO VOLUME and TO DISK clause](#) for tables in the MergeTree family:

Returns Ok. even if table does not exist. Returns error when database does not exist:

```
SYSTEM START MOVES [[db.]merge_tree_family_table_name]
```

# Managing ReplicatedMergeTree Tables

ClickHouse can manage background replication related processes in [ReplicatedMergeTree](#) tables.

## STOP FETCHES

Provides possibility to stop background fetches for inserted parts for tables in the [ReplicatedMergeTree](#) family:

Always returns Ok. regardless of the table engine and even if table or database does not exist.

```
SYSTEM STOP FETCHES [[db.]replicated_merge_tree_family_table_name]
```

## START FETCHES

Provides possibility to start background fetches for inserted parts for tables in the [ReplicatedMergeTree](#) family:

Always returns Ok. regardless of the table engine and even if table or database does not exist.

```
SYSTEM START FETCHES [[db.]replicated_merge_tree_family_table_name]
```

## STOP REPLICATED SENDS

Provides possibility to stop background sends to other replicas in cluster for new inserted parts for tables in the [ReplicatedMergeTree](#) family:

```
SYSTEM STOP REPLICATED SENDS [[db.]replicated_merge_tree_family_table_name]
```

## START REPLICATED SENDS

Provides possibility to start background sends to other replicas in cluster for new inserted parts for tables in the [ReplicatedMergeTree](#) family:

```
SYSTEM START REPLICATED SENDS [[db.]replicated_merge_tree_family_table_name]
```

## STOP REPLICATION QUEUES

Provides possibility to stop background fetch tasks from replication queues which stored in Zookeeper for tables in the [ReplicatedMergeTree](#) family. Possible background tasks types - merges, fetches, mutation, DDL statements with ON CLUSTER clause:

```
SYSTEM STOP REPLICATION QUEUES [[db.]replicated_merge_tree_family_table_name]
```

## START REPLICATION QUEUES

Provides possibility to start background fetch tasks from replication queues which stored in Zookeeper for tables in the ReplicatedMergeTree family. Possible background tasks types - merges, fetches, mutation, DDL statements with ON CLUSTER clause:

```
SYSTEM START REPLICATION QUEUES [[db.]replicated_merge_tree_family_table_name]
```

## SYNC REPLICA

Wait until a ReplicatedMergeTree table will be synced with other replicas in a cluster. Will run until receive\_timeout if fetches currently disabled for the table.

```
SYSTEM SYNC REPLICA [db.]replicated_merge_tree_family_table_name
```

After running this statement the [db.]replicated\_merge\_tree\_family\_table\_name fetches commands from the common replicated log into its own replication queue, and then the query waits till the replica processes all of the fetched commands.

## RESTART REPLICA

Provides possibility to reinitialize Zookeeper sessions state for ReplicatedMergeTree table, will compare current state with Zookeeper as source of true and add tasks to Zookeeper queue if needed.

Initialization replication queue based on ZooKeeper date happens in the same way as ATTACH TABLE statement. For a short time the table will be unavailable for any operations.

```
SYSTEM RESTART REPLICA [db.]replicated_merge_tree_family_table_name
```

## RESTORE REPLICA

Restores a replica if data is [possibly] present but Zookeeper metadata is lost.

Works only on readonly ReplicatedMergeTree tables.

One may execute query after:

- ZooKeeper root / loss.
- Replicas path /replicas loss.
- Individual replica path /replicas/*replica\_name*/ loss.

Replica attaches locally found parts and sends info about them to Zookeeper.

Parts present on a replica before metadata loss are not re-fetched from other ones if not being outdated (so replica restoration does not mean re-downloading all data over the network).

### Warning

Parts in all states are moved to `detached/` folder. Parts active before data loss (committed) are attached.

## Syntax

```
SYSTEM RESTORE REPLICA [db.]replicated_merge_tree_family_table_name [ON CLUSTER cluster_name]
```

Alternative syntax:

```
SYSTEM RESTORE REPLICA [ON CLUSTER cluster_name] [db.]replicated_merge_tree_family_table_name
```

## Example

Creating a table on multiple servers. After the replica's metadata in ZooKeeper is lost, the table will attach as read-only as metadata is missing. The last query needs to execute on every replica.

```
CREATE TABLE test(n UInt32)
ENGINE = ReplicatedMergeTree('/clickhouse/tables/test/', '{replica}')
ORDER BY n PARTITION BY n % 10;

INSERT INTO test SELECT * FROM numbers(1000);

-- zookeeper_delete_path("/clickhouse/tables/test", recursive=True) <- root loss.

SYSTEM RESTART REPLICA test;
SYSTEM RESTORE REPLICA test;
```

Another way:

```
SYSTEM RESTORE REPLICA test ON CLUSTER cluster;
```

## RESTART REPLICAS

Provides possibility to reinitialize Zookeeper sessions state for all ReplicatedMergeTree tables, will compare current state with Zookeeper as source of true and add tasks to Zookeeper queue if needed

## SHOW Statements

### SHOW CREATE TABLE

```
SHOW CREATE [TEMPORARY] [TABLE|DICTIONARY|VIEW] [db.]table|view [INTO OUTFILE filename] [FORMAT format]
```

Returns a single String-type 'statement' column, which contains a single value – the `CREATE` query used for creating the specified object.

### SHOW DATABASES

Prints a list of all databases.

```
SHOW DATABASES [LIKE | ILIKE | NOT LIKE '<pattern>'] [LIMIT <N>] [INTO OUTFILE filename] [FORMAT format]
```

This statement is identical to the query:

```
SELECT name FROM system.databases [WHERE name LIKE | ILIKE | NOT LIKE '<pattern>'] [LIMIT <N>] [INTO OUTFILE filename] [FORMAT format]
```

## Examples

Getting database names, containing the symbols sequence 'de' in their names:

```
SHOW DATABASES LIKE '%de%'
```

Result:

name
default

Getting database names, containing symbols sequence 'de' in their names, in the case insensitive manner:

```
SHOW DATABASES ILIKE '%DE%'
```

Result:

name
default

Getting database names, not containing the symbols sequence 'de' in their names:

```
SHOW DATABASES NOT LIKE '%de%'
```

Result:

name
_temporary_and_external_tables
system
test
tutorial

Getting the first two rows from database names:

```
SHOW DATABASES LIMIT 2
```

Result:

name
_temporary_and_external_tables
default

## See Also

- [CREATE DATABASE](#)

## SHOW PROCESSLIST

```
SHOW PROCESSLIST [INTO OUTFILE filename] [FORMAT format]
```

Outputs the content of the [system.processes](#) table, that contains a list of queries that is being processed at the moment, excepting `SHOW PROCESSLIST` queries.

The `SELECT * FROM system.processes` query returns data about all the current queries.

Tip (execute in the console):

```
$ watch -n1 "clickhouse-client --query='SHOW PROCESSLIST'"
```

## SHOW TABLES

Displays a list of tables.

```
SHOW [TEMPORARY] TABLES [{FROM | IN} <db>] [LIKE | ILIKE | NOT LIKE '<pattern>'] [LIMIT <N>] [INTO OUTFILE <filename>] [FORMAT <format>]
```

If the `FROM` clause is not specified, the query returns the list of tables from the current database.

This statement is identical to the query:

```
SELECT name FROM system.tables [WHERE name LIKE | ILIKE | NOT LIKE '<pattern>'] [LIMIT <N>] [INTO OUTFILE <filename>] [FORMAT <format>]
```

## Examples

Getting table names, containing the symbols sequence 'user' in their names:

```
SHOW TABLES FROM system LIKE '%user%'
```

Result:

name
user_directories
users

Getting table names, containing sequence 'user' in their names, in the case insensitive manner:

```
SHOW TABLES FROM system ILIKE '%USER%'
```

Result:

name
user_directories
users

Getting table names, not containing the symbol sequence 's' in their names:

```
SHOW TABLES FROM system NOT LIKE '%s%'
```

Result:

name
metric_log
metric_log_0
metric_log_1

Getting the first two rows from table names:

```
SHOW TABLES FROM system LIMIT 2
```

Result:

name
aggregate_function_combinators
asynchronous_metric_log

## See Also

- [Create Tables](#)
- [SHOW CREATE TABLE](#)

## SHOW DICTIONARIES

Displays a list of [external dictionaries](#).

```
SHOW DICTIONARIES [FROM <db>] [LIKE '<pattern>'] [LIMIT <N>] [INTO OUTFILE <filename>] [FORMAT <format>]
```

If the `FROM` clause is not specified, the query returns the list of dictionaries from the current database.

You can get the same results as the `SHOW DICTIONARIES` query in the following way:

```
SELECT name FROM system.dictionaries WHERE database = <db> [AND name LIKE <pattern>] [LIMIT <N>] [INTO OUTFILE <filename>] [FORMAT <format>]
```

### Example

The following query selects the first two rows from the list of tables in the `system` database, whose names contain `reg`.

```
SHOW DICTIONARIES FROM db LIKE '%reg%' LIMIT 2
```

name
regions
region_names

## SHOW GRANTS

Shows privileges for a user.

### Syntax

```
SHOW GRANTS [FOR user1 [, user2 ...]]
```

If user is not specified, the query returns privileges for the current user.

## SHOW CREATE USER

Shows parameters that were used at a [user creation](#).

SHOW CREATE USER does not output user passwords.

## Syntax

```
SHOW CREATE USER [name1 [, name2 ...] | CURRENT_USER]
```

## SHOW CREATE ROLE

Shows parameters that were used at a [role creation](#).

## Syntax

```
SHOW CREATE ROLE name1 [, name2 ...]
```

## SHOW CREATE ROW POLICY

Shows parameters that were used at a [row policy creation](#).

## Syntax

```
SHOW CREATE [ROW] POLICY name ON [database1.]table1 [, [database2.]table2 ...]
```

## SHOW CREATE QUOTA

Shows parameters that were used at a [quota creation](#).

## Syntax

```
SHOW CREATE QUOTA [name1 [, name2 ...] | CURRENT]
```

## SHOW CREATE SETTINGS PROFILE

Shows parameters that were used at a [settings profile creation](#).

## Syntax

```
SHOW CREATE [SETTINGS] PROFILE name1 [, name2 ...]
```

## SHOW USERS

Returns a list of [user account](#) names. To view user accounts parameters, see the system table [system.users](#).

## Syntax

```
SHOW USERS
```

## SHOW ROLES

Returns a list of [roles](#). To view another parameters, see system tables [system.roles](#) and [system.role-grants](#).

## Syntax

```
SHOW [CURRENT|ENABLED] ROLES
```

## SHOW PROFILES

Returns a list of [setting profiles](#). To view user accounts parameters, see the system table [settings\\_profiles](#).

### Syntax

```
SHOW [SETTINGS] PROFILES
```

## SHOW POLICIES

Returns a list of [row policies](#) for the specified table. To view user accounts parameters, see the system table [system.row\\_policies](#).

### Syntax

```
SHOW [ROW] POLICIES [ON [db.]table]
```

## SHOW QUOTAS

Returns a list of [quotas](#). To view quotas parameters, see the system table [system.quotas](#).

### Syntax

```
SHOW QUOTAS
```

## SHOW QUOTA

Returns a [quota](#) consumption for all users or for current user. To view another parameters, see system tables [system.quotas\\_usage](#) and [system.quota\\_usage](#).

### Syntax

```
SHOW [CURRENT] QUOTA
```

## SHOW ACCESS

Shows all [users](#), [roles](#), [profiles](#), etc. and all their [grants](#).

### Syntax

```
SHOW ACCESS
```

## SHOW CLUSTER(s)

Returns a list of clusters. All available clusters are listed in the [system.clusters](#) table.

### Note

SHOW CLUSTER name [query](#) displays the contents of [system.clusters](#) table for this cluster.

## Syntax

```
SHOW CLUSTER '<name>'  
SHOW CLUSTERS [LIKE|NOT LIKE '<pattern>'] [LIMIT <N>]
```

## Examples

Query:

```
SHOW CLUSTERS;
```

Result:

```
cluster  
+-----+  
| test_cluster_two_shards |  
+-----+  
| test_cluster_two_shards_internal_replication |  
+-----+  
| test_cluster_two_shards_localhost |  
+-----+  
| test_shard_localhost |  
+-----+  
| test_shard_localhost_secure |  
+-----+  
| test_unavailable_shard |
```

Query:

```
SHOW CLUSTERS LIKE 'test%' LIMIT 1;
```

Result:

```
cluster  
+-----+  
| test_cluster_two_shards |
```

Query:

```
SHOW CLUSTER 'test_shard_localhost' FORMAT Vertical;
```

Result:

```
Row 1:  
_____  
cluster: test_shard_localhost  
shard_num: 1  
shard_weight: 1  
replica_num: 1  
host_name: localhost  
host_address: 127.0.0.1  
port: 9000  
is_local: 1  
user: default  
default_database:  
errors_count: 0  
estimated_recovery_time: 0
```

## SHOW SETTINGS

Returns a list of system settings and their values. Selects data from the `system.settings` table.

## Syntax

```
SHOW [CHANGED] SETTINGS LIKE|ILIKE <name>
```

## Clauses

LIKE|ILIKE allow to specify a matching pattern for the setting name. It can contain globs such as % or \_ . LIKE clause is case-sensitive, ILIKE — case insensitive.

When the CHANGED clause is used, the query returns only settings changed from their default values.

## Examples

Query with the LIKE clause:

```
SHOW SETTINGS LIKE 'send_timeout';
```

Result:

name	type	value
send_timeout	Seconds	300

Query with the ILIKE clause:

```
SHOW SETTINGS ILIKE '%CONNECT_timeout%'
```

Result:

name	type	value
connect_timeout	Seconds	10
connect_timeout_with_failover_ms	Milliseconds	50
connect_timeout_with_failover_secure_ms	Milliseconds	100

Query with the CHANGED clause:

```
SHOW CHANGED SETTINGS ILIKE '%MEMORY%'
```

Result:

name	type	value
max_memory_usage	UInt64	100000000000

## See Also

- [system.settings](#) table

# GRANT Statement

- 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] [WITH REPLACE 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.

The **WITH REPLACE OPTION** clause replace old privileges by new privileges for the **user** or **role**, if not specified it is append privileges.

## Assigning Role Syntax

```
GRANT [ON CLUSTER cluster_name] role [...] TO {user | another_role | CURRENT_USER} [...] [WITH ADMIN OPTION] [WITH REPLACE OPTION]
```

- **role** — ClickHouse user role.
- **user** — ClickHouse user account.

The **WITH ADMIN OPTION** clause grants **ADMIN OPTION** privilege to **user** or **role**.

The **WITH REPLACE OPTION** clause replace old roles by new role for the **user** or **role**, if not specified it is append roles.

## 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 does not 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 SAMPLE 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 TEMPORARY TABLE
  - CREATE VIEW
  - CREATE DICTIONARY
  - CREATE FUNCTION

- **DROP**
  - `DROP DATABASE`
  - `DROP TABLE`
  - `DROP VIEW`
  - `DROP DICTIONARY`
  - `DROP FUNCTION`
- **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\_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 RELOAD FUNCTION
  - SYSTEM RELOAD FUNCTIONS
- 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.
- 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 does not 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 does not 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 SAMPLE BY**. Level: TABLE. Aliases: ALTER MODIFY SAMPLE BY, MODIFY SAMPLE 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: ALTER FETCH PART, FETCH PARTITION, FETCH PART
  - **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 does not 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 TEMPORARY TABLE`. Level: GLOBAL
  - `CREATE VIEW`. Level: VIEW
  - `CREATE DICTIONARY`. Level: DICTIONARY

### 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: GROUP
  - `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
  - **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 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.

# EXPLAIN Statement

Shows the execution plan of a statement.

Syntax:

```
EXPLAIN [AST | SYNTAX | PLAN | PIPELINE] [setting = value, ...] SELECT ... [FORMAT ...]
```

Example:

```
EXPLAIN SELECT sum(number) FROM numbers(10) UNION ALL SELECT sum(number) FROM numbers(10) ORDER BY sum(number) ASC FORMAT TSV;
```

```
Union
Expression (Projection)
  Expression (Before ORDER BY and SELECT)
    Aggregating
      Expression (Before GROUP BY)
        SettingQuotaAndLimits (Set limits and quota after reading from storage)
          ReadFromStorage (SystemNumbers)
Expression (Projection)
  MergingSorted (Merge sorted streams for ORDER BY)
    MergeSorting (Merge sorted blocks for ORDER BY)
      PartialSorting (Sort each block for ORDER BY)
        Expression (Before ORDER BY and SELECT)
          Aggregating
            Expression (Before GROUP BY)
              SettingQuotaAndLimits (Set limits and quota after reading from storage)
                ReadFromStorage (SystemNumbers)
```

## EXPLAIN Types

- **AST** — Abstract syntax tree.
- **SYNTAX** — Query text after AST-level optimizations.
- **PLAN** — Query execution plan.
- **PIPELINE** — Query execution pipeline.

### EXPLAIN AST

Dump query AST. Supports all types of queries, not only `SELECT`.

Examples:

```
EXPLAIN AST SELECT 1;
```

```
SelectWithUnionQuery (children 1)
  ExpressionList (children 1)
    SelectQuery (children 1)
      ExpressionList (children 1)
        Literal UInt64_1
```

```
EXPLAIN AST ALTER TABLE t1 DELETE WHERE date = today();
```

```
explain
AlterQuery t1 (children 1)
ExpressionList (children 1)
AlterCommand 27 (children 1)
Function equals (children 1)
ExpressionList (children 2)
Identifier date
Function today (children 1)
ExpressionList
```

## EXPLAIN SYNTAX

Returns query after syntax optimizations.

Example:

```
EXPLAIN SYNTAX SELECT * FROM system.numbers AS a, system.numbers AS b, system.numbers AS c;
```

```
SELECT
`--a.number` AS `a.number`,
`--b.number` AS `b.number`,
number AS `c.number`
FROM
(
  SELECT
    number AS `--a.number`,
    b.number AS `--b.number`
  FROM system.numbers AS a
  CROSS JOIN system.numbers AS b
) AS `--s`
CROSS JOIN system.numbers AS c
```

## EXPLAIN PLAN

Dump query plan steps.

Settings:

- `header` — Prints output header for step. Default: 0.
- `description` — Prints step description. Default: 1.
- `indexes` — Shows used indexes, the number of filtered parts and the number of filtered granules for every index applied. Default: 0. Supported for **MergeTree** tables.
- `actions` — Prints detailed information about step actions. Default: 0.
- `json` — Prints query plan steps as a row in **JSON** format. Default: 0. It is recommended to use **TSVRaw** format to avoid unnecessary escaping.

Example:

```
EXPLAIN SELECT sum(number) FROM numbers(10) GROUP BY number % 4;
```

```
Union
Expression (Projection)
Expression (Before ORDER BY and SELECT)
Aggregating
Expression (Before GROUP BY)
SettingQuotaAndLimits (Set limits and quota after reading from storage)
ReadFromStorage (SystemNumbers)
```

## Note

Step and query cost estimation is not supported.

When `json = 1`, the query plan is represented in JSON format. Every node is a dictionary that always has the keys `Node Type` and `Plans`. `Node Type` is a string with a step name. `Plans` is an array with child step descriptions. Other optional keys may be added depending on node type and settings.

Example:

```
EXPLAIN json = 1, description = 0 SELECT 1 UNION ALL SELECT 2 FORMAT TSVRaw;
```

```
[  
  {  
    "Plan": {  
      "Node Type": "Union",  
      "Plans": [  
        {  
          "Node Type": "Expression",  
          "Plans": [  
            {  
              "Node Type": "SettingQuotaAndLimits",  
              "Plans": [  
                {  
                  "Node Type": "ReadFromStorage"  
                }  
              ]  
            }  
          ]  
        },  
        {  
          "Node Type": "Expression",  
          "Plans": [  
            {  
              "Node Type": "SettingQuotaAndLimits",  
              "Plans": [  
                {  
                  "Node Type": "ReadFromStorage"  
                }  
              ]  
            }  
          ]  
        }  
      ]  
    }  
  ]
```

With `description = 1`, the `Description` key is added to the step:

```
{  
  "Node Type": "ReadFromStorage",  
  "Description": "SystemOne"  
}
```

With `header = 1`, the `Header` key is added to the step as an array of columns.

Example:

```
EXPLAIN json = 1, description = 0, header = 1 SELECT 1, 2 + dummy;
```

```
[
  {
    "Plan": {
      "Node Type": "Expression",
      "Header": [
        {
          "Name": "1",
          "Type": "UInt8"
        },
        {
          "Name": "plus(2, dummy)",
          "Type": "UInt16"
        }
      ],
      "Plans": [
        {
          "Node Type": "SettingQuotaAndLimits",
          "Header": [
            {
              "Name": "dummy",
              "Type": "UInt8"
            }
          ],
          "Plans": [
            {
              "Node Type": "ReadFromStorage",
              "Header": [
                {
                  "Name": "dummy",
                  "Type": "UInt8"
                }
              ]
            }
          ]
        }
      ]
    }
  ]
]
```

With `indexes = 1`, the `Indexes` key is added. It contains an array of used indexes. Each index is described as JSON with `Type` key (a string `MinMax`, `Partition`, `PrimaryKey` or `Skip`) and optional keys:

- `Name` — An index name (for now, is used only for `Skip` index).
- `Keys` — An array of columns used by the index.
- `Condition` — A string with condition used.
- `Description` — An index (for now, is used only for `Skip` index).
- `Initial Parts` — A number of parts before the index is applied.
- `Selected Parts` — A number of parts after the index is applied.
- `Initial Granules` — A number of granules before the index is applied.
- `Selected Granules` — A number of granules after the index is applied.

Example:

```

"Node Type": "ReadFromMergeTree",
"Indexes": [
{
  "Type": "MinMax",
  "Keys": ["y"],
  "Condition": "(y in [1, +inf))",
  "Initial Parts": 5,
  "Selected Parts": 4,
  "Initial Granules": 12,
  "Selected Granules": 11
},
{
  "Type": "Partition",
  "Keys": ["y", "bitAnd(z, 3)"],
  "Condition": "and((bitAnd(z, 3) not in [1, 1]), and((y in [1, +inf)), (bitAnd(z, 3) not in [1, 1])))",
  "Initial Parts": 4,
  "Selected Parts": 3,
  "Initial Granules": 11,
  "Selected Granules": 10
},
{
  "Type": "PrimaryKey",
  "Keys": ["x", "y"],
  "Condition": "and((x in [11, +inf)), (y in [1, +inf)))",
  "Initial Parts": 3,
  "Selected Parts": 2,
  "Initial Granules": 10,
  "Selected Granules": 6
},
{
  "Type": "Skip",
  "Name": "t_minmax",
  "Description": "minmax GRANULARITY 2",
  "Initial Parts": 2,
  "Selected Parts": 1,
  "Initial Granules": 6,
  "Selected Granules": 2
},
{
  "Type": "Skip",
  "Name": "t_set",
  "Description": "set GRANULARITY 2",
  "Initial Parts": 1,
  "Selected Parts": 1,
  "Initial Granules": 2,
  "Selected Granules": 1
}
]

```

With `actions = 1`, added keys depend on step type.

Example:

```
EXPLAIN json = 1, actions = 1, description = 0 SELECT 1 FORMAT TSVRaw;
```

```
[
  {
    "Plan": {
      "Node Type": "Expression",
      "Expression": {
        "Inputs": [],
        "Actions": [
          {
            "Node Type": "Column",
            "Result Type": "UInt8",
            "Result Type": "Column",
            "Column": "Const(UInt8)",
            "Arguments": [],
            "Removed Arguments": [],
            "Result": 0
          }
        ],
        "Outputs": [
          {
            "Name": "1",
            "Type": "UInt8"
          }
        ],
        "Positions": [0],
        "Project Input": true
      },
      "Plans": [
        {
          "Node Type": "SettingQuotaAndLimits",
          "Plans": [
            {
              "Node Type": "ReadFromStorage"
            }
          ]
        }
      ]
    }
  }
]
```

## EXPLAIN PIPELINE

Settings:

- `header` — Prints header for each output port. Default: 0.
- `graph` — Prints a graph described in the [DOT](#) graph description language. Default: 0.
- `compact` — Prints graph in compact mode if `graph` setting is enabled. Default: 1.

Example:

```
EXPLAIN PIPELINE SELECT sum(number) FROM numbers_mt(100000) GROUP BY number % 4;
```

```
(Union)
(Expression)
ExpressionTransform
(Expression)
ExpressionTransform
(Aggregating)
Resize 2 → 1
AggregatingTransform × 2
(Expression)
ExpressionTransform × 2
(SettingQuotaAndLimits)
(ReadFromStorage)
NumbersMt × 2 0 → 1
```

## EXPLAIN ESTIMATE

Shows the estimated number of rows, marks and parts to be read from the tables while processing the query. Works with tables in the **MergeTree** family.

### Example

Creating a table:

```
CREATE TABLE ttt (i Int64) ENGINE = MergeTree() ORDER BY i SETTINGS index_granularity = 16, write_final_mark = 0;
INSERT INTO ttt SELECT number FROM numbers(128);
OPTIMIZE TABLE ttt;
```

Query:

```
EXPLAIN ESTIMATE SELECT * FROM ttt;
```

Result:

database	table	parts	rows	marks
default	ttt	1	128	8

## REVOKE Statement

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;
```

## ATTACH Statement

Attaches a table or a dictionary, for example, when moving a database to another server.

### Syntax

```
ATTACH TABLE|DICTIONARY [IF NOT EXISTS] [db.]name [ON CLUSTER cluster] ...
```

The query does not create data on the disk, but assumes that data is already in the appropriate places, and just adds information about the table or the dictionary to the server. After executing the ATTACH query, the server will know about the existence of the table or the dictionary.

If a table was previously detached (**DETACH** query), meaning that its structure is known, you can use shorthand without defining the structure.

## Attach Existing Table

### Syntax

```
ATTACH TABLE [IF NOT EXISTS] [db.]name [ON CLUSTER cluster]
```

This query is used when starting the server. The server stores table metadata as files with ATTACH queries, which it simply runs at launch (with the exception of some system tables, which are explicitly created on the server).

If the table was detached permanently, it won't be reattached at the server start, so you need to use ATTACH query explicitly.

## Create New Table And Attach Data

### With Specified Path to Table Data

The query creates a new table with provided structure and attaches table data from the provided directory in `user_files`.

### Syntax

```
ATTACH TABLE name FROM 'path/to/data/' (col1 Type1, ...)
```

### Example

Query:

```
DROP TABLE IF EXISTS test;
INSERT INTO TABLE FUNCTION file('01188_attach/test/data.TSV', 'TSV', 's String, n UInt8') VALUES ('test', 42);
ATTACH TABLE test FROM '01188_attach/test' (s String, n UInt8) ENGINE = File(TSV);
SELECT * FROM test;
```

Result:

s	n
test	42

## With Specified Table UUID

This query creates a new table with provided structure and attaches data from the table with the specified UUID.

It is supported by the [Atomic](#) database engine.

### Syntax

```
ATTACH TABLE name UUID '<uuid>' (col1 Type1, ...)
```

## Attach Existing Dictionary

Attaches a previously detached dictionary.

### Syntax

```
ATTACH DICTIONARY [IF NOT EXISTS] [db.]name [ON CLUSTER cluster]
```

## CHECK TABLE Statement

Checks if the data in the table is corrupted.

```
CHECK TABLE [db.]name
```

The `CHECK TABLE` query compares actual file sizes with the expected values which are stored on the server. If the file sizes do not match the stored values, it means the data is corrupted. This can be caused, for example, by a system crash during query execution.

The query response contains the `result` column with a single row. The row has a value of [Boolean](#) type:

- 0 - The data in the table is corrupted.
- 1 - The data maintains integrity.

The `CHECK TABLE` query supports the following table engines:

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

Performed over the tables with another table engines causes an exception.

Engines from the `*Log` family do not provide automatic data recovery on failure. Use the `CHECK TABLE` query to track data loss in a timely manner.

## Checking the MergeTree Family Tables

For MergeTree family engines, if `check_query_single_value_result` = 0, the `CHECK TABLE` query shows a check status for every individual data part of a table on the local server.

```
SET check_query_single_value_result = 0;
CHECK TABLE test_table;
```

part_path	is_passed	message
all_1_4_1	1	
all_1_4_2	1	

If `check_query_single_value_result` = 0, the `CHECK TABLE` query shows the general table check status.

```
SET check_query_single_value_result = 1;
CHECK TABLE test_table;
```

result
1

## If the Data Is Corrupted

If the table is corrupted, you can copy the non-corrupted data to another table. To do this:

1. Create a new table with the same structure as damaged table. To do this execute the query `CREATE TABLE <new_table_name> AS <damaged_table_name>.`.
2. Set the `max_threads` value to 1 to process the next query in a single thread. To do this run the query `SET max_threads = 1.`
3. Execute the query `INSERT INTO <new_table_name> SELECT * FROM <damaged_table_name>.` This request copies the non-corrupted data from the damaged table to another table. Only the data before the corrupted part will be copied.
4. Restart the `clickhouse-client` to reset the `max_threads` value.

## Miscellaneous Statements

- [ATTACH](#)
- [CHECK TABLE](#)
- [DESCRIBE TABLE](#)
- [DETACH](#)
- [DROP](#)
- [EXISTS](#)
- [KILL](#)

- [OPTIMIZE](#)
- [RENAME](#)
- [SET](#)
- [SET ROLE](#)
- [TRUNCATE](#)
- [USE](#)

## DESCRIBE TABLE

Returns information about table columns.

### Syntax

```
DESC|DESCRIBE TABLE [db.]table [INTO OUTFILE filename] [FORMAT format]
```

The `DESCRIBE` statement returns a row for each table column with the following [String](#) values:

- `name` — A column name.
- `type` — A column type.
- `default_type` — A clause that is used in the column [default expression](#): `DEFAULT`, `MATERIALIZED` or `ALIAS`. If there is no default expression, then empty string is returned.
- `default_expression` — An expression specified after the `DEFAULT` clause.
- `comment` — A [column comment](#).
- `codec_expression` — A [codec](#) that is applied to the column.
- `ttl_expression` — A [TTL](#) expression.
- `is_subcolumn` — A flag that equals `1` for internal subcolumns. It is included into the result only if subcolumn description is enabled by the [describe\\_include\\_subcolumns](#) setting.

All columns in [Nested](#) data structures are described separately. The name of each column is prefixed with a parent column name and a dot.

To show internal subcolumns of other data types, use the [describe\\_include\\_subcolumns](#) setting.

### Example

Query:

```
CREATE TABLE describe_example (
    id UInt64, text String DEFAULT 'unknown' CODEC(ZSTD),
    user Tuple (name String, age UInt8)
) ENGINE = MergeTree() ORDER BY id;

DESCRIBE TABLE describe_example;
DESCRIBE TABLE describe_example SETTINGS describe_include_subcolumns=1;
```

Result:

name	type	default_type	default_expression	comment	codec_expression
	ttl_expression				
id	UInt64	DEFAULT	'unknown'		ZSTD(1)
text	String				
user	Tuple(name String, age UInt8)				

The second query additionally shows subcolumns:

name	type	default_type	default_expression	comment	codec_expre
ssession	ttl_expression	is_subcolumn			
id	UInt64	DEFAULT	'unknown'		ZSTD(1)
text	String				
user	Tuple(name String, age UInt8)				
user.name	String			0	
user.age	UInt8			0	

## See Also

- [describe\\_include\\_subcolumns](#) setting.

# DETACH Statement

Makes the server "forget" about the existence of a table, a materialized view, or a dictionary.

## Syntax

```
DETACH TABLE|VIEW|DICTIONARY [IF EXISTS] [db.]name [ON CLUSTER cluster] [PERMANENTLY]
```

Detaching does not delete the data or metadata of a table, a materialized view or a dictionary. If an entity was not detached PERMANENTLY, on the next server launch the server will read the metadata and recall the table/view/dictionary again. If an entity was detached PERMANENTLY, there will be no automatic recall.

Whether a table or a dictionary was detached permanently or not, in both cases you can reattach them using the [ATTACH](#) query.

System log tables can be also attached back (e.g. `query_log`, `text_log`, etc). Other system tables can't be reattached. On the next server launch the server will recall those tables again.

`ATTACH MATERIALIZED VIEW` does not work with short syntax (without `SELECT`), but you can attach it using the `ATTACH TABLE` query.

Note that you can not detach permanently the table which is already detached (temporary). But you can attach it back and then detach permanently again.

Also you can not [DROP](#) the detached table, or [CREATE TABLE](#) with the same name as detached permanently, or replace it with the other table with [RENAME TABLE](#) query.

## Example

Creating a table:

Query:

```
CREATE TABLE test ENGINE = Log AS SELECT * FROM numbers(10);
SELECT * FROM test;
```

Result:

number
0
1
2
3
4
5
6
7
8
9

Detaching the table:

Query:

```
DETACH TABLE test;
SELECT * FROM test;
```

Result:

```
Received exception from server (version 21.4.1):
Code: 60. DB::Exception: Received from localhost:9000. DB::Exception: Table default.test does not exist.
```

## See Also

- [Materialized View](#)
- [Dictionaries](#)

## DROP Statements

Deletes existing entity. If the `IF EXISTS` clause is specified, these queries do not return an error if the entity does not exist.

### DROP DATABASE

Deletes all tables inside the `db` database, then deletes the `db` database itself.

Syntax:

```
DROP DATABASE [IF EXISTS] db [ON CLUSTER cluster]
```

### DROP TABLE

Deletes the table.

Syntax:

```
DROP [TEMPORARY] TABLE [IF EXISTS] [db.]name [ON CLUSTER cluster]
```

## DROP DICTIONARY

Deletes the dictionary.

Syntax:

```
DROP DICTIONARY [IF EXISTS] [db.]name
```

## DROP USER

Deletes a user.

Syntax:

```
DROP USER [IF EXISTS] name [,....] [ON CLUSTER cluster_name]
```

## DROP ROLE

Deletes a role. The deleted role is revoked from all the entities where it was assigned.

Syntax:

```
DROP ROLE [IF EXISTS] name [,....] [ON CLUSTER cluster_name]
```

## DROP ROW POLICY

Deletes a row policy. Deleted row policy is revoked from all the entities where it was assigned.

Syntax:

```
DROP [ROW] POLICY [IF EXISTS] name [,....] ON [database.]table [,....] [ON CLUSTER cluster_name]
```

## DROP QUOTA

Deletes a quota. The deleted quota is revoked from all the entities where it was assigned.

Syntax:

```
DROP QUOTA [IF EXISTS] name [,....] [ON CLUSTER cluster_name]
```

## DROP SETTINGS PROFILE

Deletes a settings profile. The deleted settings profile is revoked from all the entities where it was assigned.

Syntax:

```
DROP [SETTINGS] PROFILE [IF EXISTS] name [,....] [ON CLUSTER cluster_name]
```

## DROP VIEW

Deletes a view. Views can be deleted by a `DROP TABLE` command as well but `DROP VIEW` checks that `[db.]name` is a view.

Syntax:

```
DROP VIEW [IF EXISTS] [db.]name [ON CLUSTER cluster]
```

## DROP FUNCTION

Deletes a user defined function created by [CREATE FUNCTION](#).

System functions can not be dropped.

### Syntax

```
DROP FUNCTION [IF EXISTS] function_name
```

### Example

```
CREATE FUNCTION linear_equation AS (x, k, b) -> k*x + b;  
DROP FUNCTION linear_equation;
```

## EXISTS Statement

```
EXISTS [TEMPORARY] [TABLE|DICTIONARY] [db.]name [INTO OUTFILE filename] [FORMAT format]
```

Returns a single UInt8-type column, which contains the single value 0 if the table or database does not exist, or 1 if the table exists in the specified database.

## KILL Statements

There are two kinds of kill statements: to kill a query and to kill a mutation

### KILL QUERY

```
KILL QUERY [ON CLUSTER cluster]  
WHERE <where expression to SELECT FROM system.processes query>  
[SYNC|ASYNC|TEST]  
[FORMAT format]
```

Attempts to forcibly terminate the currently running queries.

The queries to terminate are selected from the system.processes table using the criteria defined in the WHERE clause of the KILL query.

Examples:

```
-- 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
```

Read-only users can only stop their own queries.

By default, the asynchronous version of queries is used (ASYNC), which does not wait for confirmation that queries have stopped.

The synchronous version (SYNC) waits for all queries to stop and displays information about each process as it stops.

The response contains the `kill_status` column, which can take the following values:

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.

A test query (TEST) only checks the user's rights and displays a list of queries to stop.

## KILL MUTATION

```
KILL MUTATION [ON CLUSTER cluster]
  WHERE <where expression to SELECT FROM system.mutations query>
  [TEST]
  [FORMAT format]
```

Tries to cancel and remove **mutations** that are currently executing. Mutations to cancel are selected from the `system.mutations` table using the filter specified by the `WHERE` clause of the `KILL` query.

A test query (TEST) only checks the user's rights and displays a list of mutations to stop.

Examples:

```
-- 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).

Changes already made by the mutation are not rolled back.

## OPTIMIZE Statement

This query tries to initialize an unscheduled merge of data parts for tables.

### Warning

`OPTIMIZE` can't fix the `Too many parts` error.

### Syntax

```
OPTIMIZE TABLE [db.]name [ON CLUSTER cluster] [PARTITION partition | PARTITION ID 'partition_id'] [FINAL]
  [DEDUPLICATE [BY expression]]
```

The `OPTIMIZE` query is supported for **MergeTree** family, the **MaterializedView** and the **Buffer** engines. Other table engines aren't supported.

When `OPTIMIZE` is used with the **ReplicatedMergeTree** family of table engines, ClickHouse creates a task for merging and waits for execution on all replicas (if the `replication_alter_partitions_sync` setting is set to 2) or on current replica (if the `replication_alter_partitions_sync` setting is set to 1).

- If `OPTIMIZE` does not perform a merge for any reason, it does not notify the client. To enable notifications, use the `optimize_throw_if_noop` setting.
- If you specify a `PARTITION`, only the specified partition is optimized. [How to set partition expression](#).
- If you specify `FINAL`, optimization is performed even when all the data is already in one part. Also merge is forced even if concurrent merges are performed.
- If you specify `DEDUPLICATE`, then completely identical rows (unless `by`-clause is specified) will be deduplicated (all columns are compared), it makes sense only for the MergeTree engine.

You can specify how long (in seconds) to wait for inactive replicas to execute `OPTIMIZE` queries by the `replication_wait_for_inactive_replica_timeout` setting.

## Note

If the `replication_alter_partitions_sync` is set to 2 and some replicas are not active for more than the time, specified by the `replication_wait_for_inactive_replica_timeout` setting, then an exception `UNFINISHED` is thrown.

## BY expression

If you want to perform deduplication on custom set of columns rather than on all, you can specify list of columns explicitly or use any combination of `*`, `COLUMNS` or `EXCEPT` expressions. The explicitly written or implicitly expanded list of columns must include all columns specified in row ordering expression (both primary and sorting keys) and partitioning expression (partitioning key).

## Note

Notice that `*` behaves just like in `SELECT`: **MATERIALIZED** and **ALIAS** columns are not used for expansion.

Also, it is an error to specify empty list of columns, or write an expression that results in an empty list of columns, or deduplicate by an `ALIAS` column.

## Syntax

```
OPTIMIZE TABLE table DEDUPLICATE; -- all columns
OPTIMIZE TABLE table DEDUPLICATE BY *; -- excludes MATERIALIZED and ALIAS columns
OPTIMIZE TABLE table DEDUPLICATE BY colX,colY,colZ;
OPTIMIZE TABLE table DEDUPLICATE BY * EXCEPT colX;
OPTIMIZE TABLE table DEDUPLICATE BY * EXCEPT (colX, colY);
OPTIMIZE TABLE table DEDUPLICATE BY COLUMNS('column-matched-by-regex');
OPTIMIZE TABLE table DEDUPLICATE BY COLUMNS('column-matched-by-regex') EXCEPT colX;
OPTIMIZE TABLE table DEDUPLICATE BY COLUMNS('column-matched-by-regex') EXCEPT (colX, colY);
```

## Examples

Consider the table:

```

CREATE TABLE example (
    primary_key Int32,
    secondary_key Int32,
    value UInt32,
    partition_key UInt32,
    materialized_value UInt32 MATERIALIZED 12345,
    aliased_value UInt32 ALIAS 2,
    PRIMARY KEY primary_key
) ENGINE=MergeTree
PARTITION BY partition_key
ORDER BY (primary_key, secondary_key);

```

```

INSERT INTO example (primary_key, secondary_key, value, partition_key)
VALUES (0, 0, 0, 0), (0, 0, 0, 0), (1, 1, 2, 2), (1, 1, 2, 3), (1, 1, 3, 3);

```

```
SELECT * FROM example;
```

Result:

primary_key	secondary_key	value	partition_key
0	0	0	0
0	0	0	0
1	1	2	2
1	1	2	3
1	1	3	3

When columns for deduplication are not specified, all of them are taken into account. Row is removed only if all values in all columns are equal to corresponding values in previous row:

```
OPTIMIZE TABLE example FINAL DEDUPLICATE;
```

```
SELECT * FROM example;
```

Result:

primary_key	secondary_key	value	partition_key
1	1	2	2
0	0	0	0
1	1	2	3

When columns are specified implicitly, the table is deduplicated by all columns that are not ALIAS or MATERIALIZED. Considering the table above, these are `primary_key`, `secondary_key`, `value`, and `partition_key` columns:

```
OPTIMIZE TABLE example FINAL DEDUPLICATE BY *;
```

```
SELECT * FROM example;
```

Result:

primary_key	secondary_key	value	partition_key
1	1	2	2
0	0	0	0
1	1	2	3
1	1	3	3

Deduplicate by all columns that are not `ALIAS` or `MATERIALIZED` and explicitly not `value`: `primary_key`, `secondary_key`, and `partition_key` columns.

```
OPTIMIZE TABLE example FINAL DEDUPLICATE BY * EXCEPT value;
```

```
SELECT * FROM example;
```

Result:

primary_key	secondary_key	value	partition_key
1	1	2	2
0	0	0	0
1	1	2	3

Deduplicate explicitly by `primary_key`, `secondary_key`, and `partition_key` columns:

```
OPTIMIZE TABLE example FINAL DEDUPLICATE BY primary_key, secondary_key, partition_key;
```

```
SELECT * FROM example;
```

Result:

primary_key	secondary_key	value	partition_key
1	1	2	2
0	0	0	0
1	1	2	3

Deduplicate by any column matching a regex: `primary_key`, `secondary_key`, and `partition_key` columns:

```
OPTIMIZE TABLE example FINAL DEDUPLICATE BY COLUMNS('.*_key');
```

```
SELECT * FROM example;
```

Result:

primary_key	secondary_key	value	partition_key
0	0	0	0
1	1	2	2
1	1	2	3

## RENAME Statement

Renames databases, tables, or dictionaries. Several entities can be renamed in a single query. Note that the `RENAME` query with several entities is non-atomic operation. To swap entities names atomically, use the `EXCHANGE` statement.

### Note

The `RENAME` query is supported by the **Atomic** database engine only.

### Syntax

```
RENAME DATABASE|TABLE|DICTIONARY name TO new_name [...] [ON CLUSTER cluster]
```

## RENAME DATABASE

Renames databases.

### Syntax

```
RENAME DATABASE atomic_database1 TO atomic_database2 [...] [ON CLUSTER cluster]
```

## RENAME TABLE

Renames one or more tables.

Renaming tables is a light operation. If you pass a different database after `TO`, the table will be moved to this database. However, the directories with databases must reside in the same file system. Otherwise, an error is returned.

If you rename multiple tables in one query, the operation is not atomic. It may be partially executed, and queries in other sessions may get `Table ... does not exist ...` error.

### Syntax

```
RENAME TABLE [db1.]name1 TO [db2.]name2 [...] [ON CLUSTER cluster]
```

### Example

```
RENAME TABLE table_A TO table_A_bak, table_B TO table_B_bak;
```

## RENAME DICTIONARY

Renames one or several dictionaries. This query can be used to move dictionaries between databases.

### Syntax

```
RENAME DICTIONARY [db0.]dict_A TO [db1.]dict_B [,...] [ON CLUSTER cluster]
```

### See Also

- [Dictionaries](#)

## EXCHANGE Statement

Exchanges the names of two tables or dictionaries atomically.

This task can also be accomplished with a [RENAME](#) query using a temporary name, but the operation is not atomic in that case.

### Note

The EXCHANGE query is supported by the [Atomic](#) database engine only.

### Syntax

```
EXCHANGE TABLES|DICTIONARIES [db0.]name_A AND [db1.]name_B
```

## EXCHANGE TABLES

Exchanges the names of two tables.

### Syntax

```
EXCHANGE TABLES [db0.]table_A AND [db1.]table_B
```

## EXCHANGE DICTIONARIES

Exchanges the names of two dictionaries.

### Syntax

```
EXCHANGE DICTIONARIES [db0.]dict_A AND [db1.]dict_B
```

### See Also

- [Dictionaries](#)

## SET Statement

```
SET param = value
```

Assigns `value` to the `param` **setting** for the current session. You cannot change **server settings** this way.

You can also set all the values from the specified settings profile in a single query.

```
SET profile = 'profile-name-from-the-settings-file'
```

For more information, see [Settings](#).

## SET ROLE Statement

Activates roles for the current user.

```
SET ROLE {DEFAULT | NONE | role [,...] | ALL | ALL EXCEPT role [,...]}
```

### SET DEFAULT ROLE

Sets default roles to a user.

Default roles are automatically activated at user login. You can set as default only the previously granted roles. If the role isn't granted to a user, ClickHouse throws an exception.

```
SET DEFAULT ROLE {NONE | role [,...] | ALL | ALL EXCEPT role [,...]} TO {user|CURRENT_USER} [,...]
```

## Examples

Set multiple default roles to a user:

```
SET DEFAULT ROLE role1, role2, ... TO user
```

Set all the granted roles as default to a user:

```
SET DEFAULT ROLE ALL TO user
```

Purge default roles from a user:

```
SET DEFAULT ROLE NONE TO user
```

Set all the granted roles as default excepting some of them:

```
SET DEFAULT ROLE ALL EXCEPT role1, role2 TO user
```

## TRUNCATE Statement

```
TRUNCATE TABLE [IF EXISTS] [db.]name [ON CLUSTER cluster]
```

Removes all data from a table. When the clause `IF EXISTS` is omitted, the query returns an error if the table does not exist.

The `TRUNCATE` query is not supported for `View`, `File`, `URL`, `Buffer` and `Null` table engines.

You can use the `replication_alter_partitions_sync` setting to set up waiting for actions to be executed on replicas.

You can specify how long (in seconds) to wait for inactive replicas to execute `TRUNCATE` queries with the `replication_wait_for_inactive_replica_timeout` setting.

## Note

If the `replication_alter_partitions_sync` is set to 2 and some replicas are not active for more than the time, specified by the `replication_wait_for_inactive_replica_timeout` setting, then an exception `UNFINISHED` is thrown.

## USE Statement

```
USE db
```

Lets you set the current database for the session.

The current database is used for searching for tables if the database is not explicitly defined in the query with a dot before the table name.

This query can't be made when using the HTTP protocol, since there is no concept of a session.

## WATCH Statement (Experimental)

### Important

This is an experimental feature that may change in backwards-incompatible ways in the future releases.

Enable live views and `WATCH` query using `set allow_experimental_live_view = 1`.

```
WATCH [db.]live_view  
[EVENTS]  
[LIMIT n]  
[FORMAT format]
```

The `WATCH` query performs continuous data retrieval from a `LIVE VIEW` table. Unless the `LIMIT` clause is specified it provides an infinite stream of query results from a `LIVE VIEW`.

```
WATCH [db.]live_view [EVENTS] [LIMIT n] [FORMAT format]
```

## Virtual columns

The virtual `_version` column in the query result indicates the current result version.

### **Example:**

```
CREATE LIVE VIEW lv WITH REFRESH 5 AS SELECT now();
WATCH lv;
```

now()	version
2021-02-21 09:17:21	1
2021-02-21 09:17:26	2
2021-02-21 09:17:31	3
...	

By default, the requested data is returned to the client, while in conjunction with **INSERT INTO** it can be forwarded to a different table.

### **Example:**

```
INSERT INTO [db.]table WATCH [db.]live_view ...
```

## EVENTS Clause

The **EVENTS** clause can be used to obtain a short form of the **WATCH** query where instead of the query result you will just get the latest query result version.

```
WATCH [db.]live_view EVENTS;
```

### **Example:**

```
CREATE LIVE VIEW lv WITH REFRESH 5 AS SELECT now();
WATCH lv EVENTS;
```

version
1
2
...

## LIMIT Clause

The **LIMIT n** clause specifies the number of updates the **WATCH** query should wait for before terminating. By default there is no limit on the number of updates and therefore the query will not terminate. The value of 0 indicates that the **WATCH** query should not wait for any new query results and therefore will return immediately once query result is evaluated.

```
WATCH [db.]live_view LIMIT 1;
```

### **Example:**

```
CREATE LIVE VIEW lv WITH REFRESH 5 AS SELECT now();
WATCH lv EVENTS LIMIT 1;
```

```
version
1 |
```

## FORMAT Clause

The `FORMAT` clause works the same way as for the [SELECT](#).

### Note

The [JSONEachRowWithProgress](#) format should be used when watching [LIVE VIEW](#) tables over the HTTP interface. The progress messages will be added to the output to keep the long-lived HTTP connection alive until the query result changes. The interval between progress messages is controlled using the [live\\_view\\_heartbeat\\_interval](#) setting.

## Syntax

There are two types of parsers in the system: the full SQL parser (a recursive descent parser), and the data format parser (a fast stream parser).

In all cases except the `INSERT` query, only the full SQL parser is used.

The `INSERT` query uses both parsers:

```
INSERT INTO t VALUES (1, 'Hello, world'), (2, 'abc'), (3, 'def')
```

The `INSERT INTO t VALUES` fragment is parsed by the full parser, and the data `(1, 'Hello, world'), (2, 'abc'), (3, 'def')` is parsed by the fast stream parser. You can also turn on the full parser for the data by using the [input\\_format\\_values\\_interpret\\_expressions](#) setting. When `input_format_values_interpret_expressions = 1`, ClickHouse first tries to parse values with the fast stream parser. If it fails, ClickHouse tries to use the full parser for the data, treating it like an SQL [expression](#).

Data can have any format. When a query is received, the server calculates no more than [max\\_query\\_size](#) bytes of the request in RAM (by default, 1 MB), and the rest is stream parsed.

It allows for avoiding issues with large `INSERT` queries.

When using the `Values` format in an `INSERT` query, it may seem that data is parsed the same as expressions in a `SELECT` query, but this is not true. The `Values` format is much more limited.

The rest of this article covers the full parser. For more information about format parsers, see the [Formats](#) section.

## Spaces

There may be any number of space symbols between syntactical constructions (including the beginning and end of a query). Space symbols include the space, tab, line feed, CR, and form feed.

## Comments

ClickHouse supports either SQL-style and C-style comments:

- SQL-style comments start with `--` and continue to the end of the line, a space after `--` can be omitted.

- C-style are from /\* to \*/ and can be multiline, spaces are not required either.

## Keywords

Keywords are case-insensitive when they correspond to:

- SQL standard. For example, `SELECT`, `select` and `SeLeCt` are all valid.
- Implementation in some popular DBMS (MySQL or Postgres). For example, `DateTime` is the same as `datetime`.

You can check whether a data type name is case-sensitive in the `system.data_type_families` table.

In contrast to standard SQL, all other keywords (including functions names) are **case-sensitive**.

Keywords are not reserved; they are treated as such only in the corresponding context. If you use **identifiers** with the same name as the keywords, enclose them into double-quotes or backticks. For example, the query `SELECT "FROM" FROM table_name` is valid if the table `table_name` has column with the name `"FROM"`.

## Identifiers

Identifiers are:

- Cluster, database, table, partition, and column names.
- Functions.
- Data types.
- **Expression aliases**.

Identifiers can be quoted or non-quoted. The latter is preferred.

Non-quoted identifiers must match the regex `^[a-zA-Z_][0-9a-zA-Z_]*$` and can not be equal to **keywords**. Examples: `x`, `_1`, `X_y_Z123_`.

If you want to use identifiers the same as keywords or you want to use other symbols in identifiers, quote it using double quotes or backticks, for example, `"id"`, ``id``.

## Literals

There are numeric, string, compound, and `NULL` literals.

### Numeric

Numeric literal tries to be parsed:

- First, as a 64-bit signed number, using the `strtoull` function.
- If unsuccessful, as a 64-bit unsigned number, using the `strtoll` function.
- If unsuccessful, as a floating-point number using the `strtod` function.
- Otherwise, it returns an error.

Literal value has the smallest type that the value fits in.

For example, `1` is parsed as `UInt8`, but `256` is parsed as `UInt16`. For more information, see **Data types**.

Examples: `1`, `18446744073709551615`, `0xDEADBEEF`, `01`, `0.1`, `1e100`, `-1e-100`, `inf`, `nan`.

### String

Only string literals in single quotes are supported. The enclosed characters can be backslash-escaped. The following escape sequences have a corresponding special value: \b, \f, \r, \n, \t, \0, \a, \v, \xHH. In all other cases, escape sequences in the format \c, where c is any character, are converted to c. It means that you can use the sequences '\and\\'. The value will have the [String](#) type.

In string literals, you need to escape at least ' and \. Single quotes can be escaped with the single quote, literals 'It's' and 'It"s' are equal.

## Compound

Arrays are constructed with square brackets [1, 2, 3]. Tuples are constructed with round brackets (1, 'Hello, world!', 2).

Technically these are not literals, but expressions with the array creation operator and the tuple creation operator, respectively.

An array must consist of at least one item, and a tuple must have at least two items.

There's a separate case when tuples appear in the IN clause of a `SELECT` query. Query results can include tuples, but tuples can't be saved to a database (except of tables with [Memory](#) engine).

## NULL

Indicates that the value is missing.

In order to store `NULL` in a table field, it must be of the [Nullable](#) type.

Depending on the data format (input or output), `NULL` may have a different representation. For more information, see the documentation for [data formats](#).

There are many nuances to processing `NULL`. For example, if at least one of the arguments of a comparison operation is `NULL`, the result of this operation is also `NULL`. The same is true for multiplication, addition, and other operations. For more information, read the documentation for each operation.

In queries, you can check `NULL` using the [IS NULL](#) and [IS NOT NULL](#) operators and the related functions `isNull` and `isNotNull`.

## Heredoc

A [heredoc](#) is a way to define a string (often multiline), while maintaining the original formatting. A heredoc is defined as a custom string literal, placed between two \$ symbols, for example `$heredoc$`. A value between two heredocs is processed "as-is".

You can use a heredoc to embed snippets of SQL, HTML, or XML code, etc.

### Example

Query:

```
SELECT $smth$SHOW CREATE VIEW my_view$smth$;
```

Result:

```
'SHOW CREATE VIEW my_view'  
SHOW CREATE VIEW my_view |
```

## Functions

Function calls are written like an identifier with a list of arguments (possibly empty) in round brackets. In contrast to standard SQL, the brackets are required, even for an empty argument list. Example: `now()`. There are regular and aggregate functions (see the section “Aggregate functions”). Some aggregate functions can contain two lists of arguments in brackets. Example: `quantile(0.9)(x)`. These aggregate functions are called “parametric” functions, and the arguments in the first list are called “parameters”. The syntax of aggregate functions without parameters is the same as for regular functions.

## Operators

Operators are converted to their corresponding functions during query parsing, taking their priority and associativity into account.

For example, the expression `1 + 2 * 3 + 4` is transformed to `plus(plus(1, multiply(2, 3)), 4)`.

## Data Types and Database Table Engines

Data types and table engines in the `CREATE` query are written the same way as identifiers or functions. In other words, they may or may not contain an argument list in brackets. For more information, see the sections “Data types,” “Table engines,” and “`CREATE`”.

## Expression Aliases

An alias is a user-defined name for expression in a query.

`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` clause without using the `AS` keyword.

For example, ``SELECT table_name_alias.column_name FROM table_name table_name_alias``.

In the [CAST](#sql-reference-sql\_reference-functions-type\_conversion\_functions-md) 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`. Aliases should comply with the [identifiers](#) syntax.

For example, ``SELECT "table t".column_name FROM table_name AS "table t"``.

## Notes on Usage

Aliases are global for a query or subquery, and you can define an alias in any part of a query for any expression. For example, `SELECT (1 AS n) + 2, n`

Aliases are not visible in subqueries and between subqueries. For example, while executing the query `SELECT (SELECT sum(b.a) + num FROM b) - a.a AS num FROM a` ClickHouse generates the exception `Unknown identifier: num`.

If an alias is defined for the result columns in the `SELECT` clause of a subquery, these columns are visible in the outer query. For example, `SELECT n + m FROM (SELECT 1 AS n, 2 AS m)`

Be careful with aliases that are the same as column or table names. Let’s consider the following example:

```
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.

In this example, we declared table `t` with column `b`. Then, when selecting data, we defined the `sum(b)` AS `b` alias. As aliases are global, ClickHouse substituted the literal `b` in the expression `argMax(a, b)` with the expression `sum(b)`. This substitution caused the exception. You can change this default behavior by setting `prefer_column_name_to_alias` to `1`.

## Asterisk

In a `SELECT` query, an asterisk can replace the expression. For more information, see the section “`SELECT`”.

## Expressions

An expression is a function, identifier, literal, application of an operator, expression in brackets, subquery, or asterisk. It can also contain an alias.

A list of expressions is one or more expressions separated by commas.

Functions and operators, in turn, can have expressions as arguments.

# Distributed DDL Queries (ON CLUSTER Clause)

By default the `CREATE`, `DROP`, `ALTER`, and `RENAME` queries affect only the current server where they are executed. In a cluster setup, it is possible to run such queries in a distributed manner with the `ON CLUSTER` clause.

For example, the following query creates the `all_hits` Distributed table on each host in `cluster`:

```
CREATE TABLE IF NOT EXISTS all_hits ON CLUSTER cluster (p Date, i Int32) ENGINE = Distributed(cluster, default, hits)
```

In order to run these queries correctly, each host must have the same cluster definition (to simplify syncing configs, you can use substitutions from ZooKeeper). They must also connect to the ZooKeeper servers.

The local version of the query will eventually be executed on each host in the cluster, even if some hosts are currently not available.

## Warning

The order for executing queries within a single host is guaranteed.

# Functions

There are at least\* two types of functions - regular functions (they are just called “functions”) and aggregate functions. These are completely different concepts. Regular functions work as if they are applied to each row separately (for each row, the result of the function does not depend on the other rows). Aggregate functions accumulate a set of values from various rows (i.e. they depend on the entire set of rows).

In this section we discuss regular functions. For aggregate functions, see the section “Aggregate functions”.

\* - There is a third type of function that the ‘arrayJoin’ function belongs to; table functions can also be mentioned separately.\*

## Strong Typing

In contrast to standard SQL, ClickHouse has strong typing. In other words, it does not make implicit conversions between types. Each function works for a specific set of types. This means that sometimes you need to use type conversion functions.

## Common Subexpression Elimination

All expressions in a query that have the same AST (the same record or same result of syntactic parsing) are considered to have identical values. Such expressions are concatenated and executed once. Identical subqueries are also eliminated this way.

## Types of Results

All functions return a single return as the result (not several values, and not zero values). The type of result is usually defined only by the types of arguments, not by the values. Exceptions are the tupleElement function (the `a.N` operator), and the `toFixedString` function.

## Constants

For simplicity, certain functions can only work with constants for some arguments. For example, the right argument of the `LIKE` operator must be a constant.

Almost all functions return a constant for constant arguments. The exception is functions that generate random numbers.

The ‘now’ function returns different values for queries that were run at different times, but the result is considered a constant, since constancy is only important within a single query.

A constant expression is also considered a constant (for example, the right half of the `LIKE` operator can be constructed from multiple constants).

Functions can be implemented in different ways for constant and non-constant arguments (different code is executed). But the results for a constant and for a true column containing only the same value should match each other.

## NULL Processing

Functions have the following behaviors:

- If at least one of the arguments of the function is `NULL`, the function result is also `NULL`.
- Special behavior that is specified individually in the description of each function. In the ClickHouse source code, these functions have `UseDefaultImplementationForNulls=false`.

## Constancy

Functions can't change the values of their arguments – any changes are returned as the result. Thus, the result of calculating separate functions does not depend on the order in which the functions are written in the query.

## Higher-order functions, `->` operator and `lambda(params, expr)` function

Higher-order functions can only accept lambda functions as their functional argument. To pass a lambda function to a higher-order function use `->` operator. The left side of the arrow has a formal parameter, which is any ID, or multiple formal parameters – any IDs in a tuple. The right side of the arrow has an expression that can use these formal parameters, as well as any table columns.

Examples:

```
x -> 2 * x
str -> str != Referer
```

A lambda function that accepts multiple arguments can also be passed to a higher-order function. In this case, the higher-order function is passed several arrays of identical length that these arguments will correspond to.

For some functions the first argument (the lambda function) can be omitted. In this case, identical mapping is assumed.

## SQL User Defined Functions

Custom functions from lambda expressions can be created using the **CREATE FUNCTION** statement. To delete these functions use the **DROP FUNCTION** statement.

## Executable User Defined Functions

ClickHouse can call any external executable program or script to process data. Describe such functions in a **configuration file** and add the path of that file to the main configuration in `user_defined_executable_functions_config` setting. If a wildcard symbol `*` is used in the path, then all files matching the pattern are loaded. Example:

```
<user_defined_executable_functions_config>*_function.xml</user_defined_executable_functions_config>
```

User defined function configurations are searched relative to the path specified in the `user_files_path` setting.

A function configuration contains the following settings:

- `name` - a function name.
- `command` - a command or a script to execute.
- `argument` - argument description with the `type` of an argument. Each argument is described in a separate setting.
- `format` - a **format** in which arguments are passed to the command.
- `return_type` - the type of a returned value.
- `type` - an executable type. If `type` is set to `executable` then single command is started. If it is set to `executable_pool` then a pool of commands is created.
- `max_command_execution_time` - maximum execution time in seconds for processing block of data. This setting is valid for `executable_pool` commands only. Optional. Default value is 10.

- `command_termination_timeout` - time in seconds during which a command should finish after its pipe is closed. After that time `SIGTERM` is sent to the process executing the command. This setting is valid for `executable_pool` commands only. Optional. Default value is `10`.
- `pool_size` - the size of a command pool. Optional. Default value is `16`.
- `lifetime` - the reload interval of a function in seconds. If it is set to `0` then the function is not reloaded.
- `send_chunk_header` - controls whether to send row count before sending a chunk of data to process. Optional. Default value is `false`.

The command must read arguments from `STDIN` and must output the result to `STDOUT`. The command must process arguments iteratively. That is after processing a chunk of arguments it must wait for the next chunk.

## Example

Creating `test_function` using XML configuration:

```
<functions>
  <function>
    <type>executable</type>
    <name>test_function</name>
    <return_type>UInt64</return_type>
    <argument>
      <type>UInt64</type>
    </argument>
    <argument>
      <type>UInt64</type>
    </argument>
    <format>TabSeparated</format>
    <command>cd /; clickhouse-local --input-format TabSeparated --output-format TabSeparated --structure 'x
UInt64, y UInt64' --query "SELECT x + y FROM table"</command>
    <lifetime>0</lifetime>
  </function>
</functions>
```

Query:

```
SELECT test_function(toUInt64(2), toUInt64(2));
```

Result:

```
test_function(toUInt64(2), toUInt64(2))—
  4 |
```

## Error Handling

Some functions might throw an exception if the data is invalid. In this case, the query is canceled and an error text is returned to the client. For distributed processing, when an exception occurs on one of the servers, the other servers also attempt to abort the query.

## Evaluation of Argument Expressions

In almost all programming languages, one of the arguments might not be evaluated for certain operators. This is usually the operators `&&`, `||`, and `?::`.

But in ClickHouse, arguments of functions (operators) are always evaluated. This is because entire parts of columns are evaluated at once, instead of calculating each row separately.

## Performing Functions for Distributed Query Processing

For distributed query processing, as many stages of query processing as possible are performed on remote servers, and the rest of the stages (merging intermediate results and everything after that) are performed on the requestor server.

This means that functions can be performed on different servers.

For example, in the query `SELECT f(sum(g(x))) FROM distributed_table GROUP BY h(y)`,

- if a `distributed_table` has at least two shards, the functions ‘g’ and ‘h’ are performed on remote servers, and the function ‘f’ is performed on the requestor server.
- if a `distributed_table` has only one shard, all the ‘f’, ‘g’, and ‘h’ functions are performed on this shard’s server.

The result of a function usually does not depend on which server it is performed on. However, sometimes this is important.

For example, functions that work with dictionaries use the dictionary that exists on the server they are running on.

Another example is the `hostName` function, which returns the name of the server it is running on in order to make `GROUP BY` by servers in a `SELECT` query.

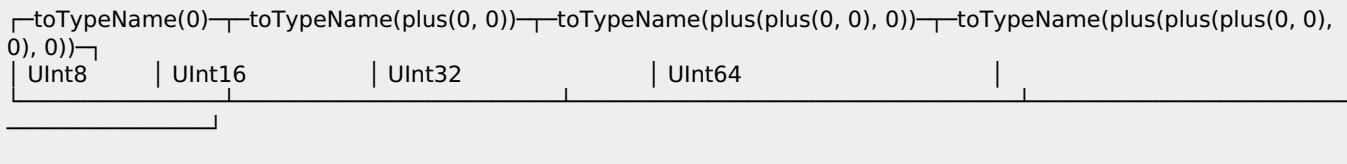
If a function in a query is performed on the requestor server, but you need to perform it on remote servers, you can wrap it in an ‘any’ aggregate function or add it to a key in `GROUP BY`.

## Arithmetic Functions

For all arithmetic functions, the result type is calculated as the smallest number type that the result fits in, if there is such a type. The minimum is taken simultaneously based on the number of bits, whether it is signed, and whether it floats. If there are not enough bits, the highest bit type is taken.

Example:

```
SELECT toTypeName(0), toTypeName(0 + 0), toTypeName(0 + 0 + 0), toTypeName(0 + 0 + 0 + 0)
```



Arithmetic functions work for any pair of types from `UInt8`, `UInt16`, `UInt32`, `UInt64`, `Int8`, `Int16`, `Int32`, `Int64`, `Float32`, or `Float64`.

Overflow is produced the same way as in C++.

### `plus(a, b)`, `a + b` operator

Calculates the sum of the numbers.

You can also add integer numbers with a date or date and time. In the case of a date, adding an integer means adding the corresponding number of days. For a date with time, it means adding the corresponding number of seconds.

### `minus(a, b)`, `a - b` operator

Calculates the difference. The result is always signed.

You can also calculate integer numbers from a date or date with time. The idea is the same – see above for ‘plus’.

## **multiply(a, b), a \* b operator**

Calculates the product of the numbers.

## **divide(a, b), a / b operator**

Calculates the quotient of the numbers. The result type is always a floating-point type.

It is not integer division. For integer division, use the ‘intDiv’ function.

When dividing by zero you get ‘inf’, ‘-inf’, or ‘nan’.

## **intDiv(a, b)**

Calculates the quotient of the numbers. Divides into integers, rounding down (by the absolute value).

An exception is thrown when dividing by zero or when dividing a minimal negative number by minus one.

## **intDivOrZero(a, b)**

Differs from ‘intDiv’ in that it returns zero when dividing by zero or when dividing a minimal negative number by minus one.

## **modulo(a, b), a % b operator**

Calculates the remainder after division.

If arguments are floating-point numbers, they are pre-converted to integers by dropping the decimal portion.

The remainder is taken in the same sense as in C++. Truncated division is used for negative numbers.

An exception is thrown when dividing by zero or when dividing a minimal negative number by minus one.

## **moduloOrZero(a, b)**

Differs from **modulo** in that it returns zero when the divisor is zero.

## **negate(a), -a operator**

Calculates a number with the reverse sign. The result is always signed.

## **abs(a)**

Calculates the absolute value of the number (a). That is, if  $a < 0$ , it returns  $-a$ . For unsigned types it does not do anything. For signed integer types, it returns an unsigned number.

## **gcd(a, b)**

Returns the greatest common divisor of the numbers.

An exception is thrown when dividing by zero or when dividing a minimal negative number by minus one.

## **lcm(a, b)**

Returns the least common multiple of the numbers.

An exception is thrown when dividing by zero or when dividing a minimal negative number by minus one.

## **max2**

Compares two values and returns the maximum. The returned value is converted to **Float64**.

## Syntax

```
max2(value1, value2)
```

## Arguments

- `value1` — First value. [Int/UInt](#) or [Float](#).
- `value2` — Second value. [Int/UInt](#) or [Float](#).

## Returned value

- The maximum of two values.

Type: [Float](#).

## Example

Query:

```
SELECT max2(-1, 2);
```

Result:

```
max2(-1, 2)
  2 |
```

## min2

Compares two values and returns the minimum. The returned value is converted to [Float64](#).

## Syntax

```
max2(value1, value2)
```

## Arguments

- `value1` — First value. [Int/UInt](#) or [Float](#).
- `value2` — Second value. [Int/UInt](#) or [Float](#).

## Returned value

- The minimum of two values.

Type: [Float](#).

## Example

Query:

```
SELECT min2(-1, 2);
```

Result:

```
└─min2(-1, 2)─  
  └─1 |
```

# Array Functions

## empty

Checks whether the input array is empty.

### Syntax

```
empty([x])
```

An array is considered empty if it does not contain any elements.

### Note

Can be optimized by enabling the **optimize\_functions\_to\_subcolumns** setting. With `optimize_functions_to_subcolumns = 1` the function reads only **size0** subcolumn instead of reading and processing the whole array column. The query `SELECT empty(arr) FROM TABLE;` transforms to `SELECT arr.size0 = 0 FROM TABLE;`.

The function also works for [strings](#) or [UUID](#).

### Arguments

- `[x]` — Input array. [Array](#).

### Returned value

- Returns `1` for an empty array or `0` for a non-empty array.

Type: [UInt8](#).

### Example

Query:

```
SELECT empty([]);
```

Result:

```
└─empty(array())─  
  └─1 |
```

## notEmpty

Checks whether the input array is non-empty.

### Syntax

```
notEmpty([x])
```

An array is considered non-empty if it contains at least one element.

## Note

Can be optimized by enabling the **optimize\_functions\_to\_subcolumns** setting. With `optimize_functions_to_subcolumns = 1` the function reads only **size0** subcolumn instead of reading and processing the whole array column. The query `SELECT notEmpty(arr) FROM table` transforms to `SELECT arr.size0 != 0 FROM TABLE`.

The function also works for **strings** or **UUID**.

### Arguments

- `[x]` — Input array. **Array**.

### Returned value

- Returns `1` for a non-empty array or `0` for an empty array.

Type: **UInt8**.

### Example

Query:

```
SELECT notEmpty([1,2]);
```

Result:

```
notEmpty([1, 2])  
1 |
```

## length

Returns the number of items in the array.

The result type is **UInt64**.

The function also works for strings.

Can be optimized by enabling the **optimize\_functions\_to\_subcolumns** setting. With `optimize_functions_to_subcolumns = 1` the function reads only **size0** subcolumn instead of reading and processing the whole array column. The query `SELECT length(arr) FROM table` transforms to `SELECT arr.size0 FROM TABLE`.

**emptyArrayUInt8**, **emptyArrayUInt16**, **emptyArrayUInt32**,  
**emptyArrayUInt64**

**emptyArrayInt8**, **emptyArrayInt16**, **emptyArrayInt32**,  
**emptyArrayInt64**

**emptyArrayFloat32**, **emptyArrayFloat64**

## emptyArrayDate, emptyArrayDateTime

### emptyArrayString

Accepts zero arguments and returns an empty array of the appropriate type.

## emptyArrayToSingle

Accepts an empty array and returns a one-element array that is equal to the default value.

## range(end), range([start, ] end [, step])

Returns an array of `UInt` numbers from `start` to `end - 1` by `step`.

### Syntax

```
range([start, ] end [, step])
```

### Arguments

- `start` — The first element of the array. Optional, required if `step` is used. Default value: 0. `UInt`
- `end` — The number before which the array is constructed. Required. `UInt`
- `step` — Determines the incremental step between each element in the array. Optional. Default value: 1. `UInt`

### Returned value

- Array of `UInt` numbers from `start` to `end - 1` by `step`.

### Implementation details

- All arguments must be positive values: `start`, `end`, `step` are `UInt` data types, as well as elements of the returned array.
- An exception is thrown if query results in arrays with a total length of more than number of elements specified by the `function_range_max_elements_in_block` setting.

### Examples

Query:

```
SELECT range(5), range(1, 5), range(1, 5, 2);
```

Result:

```
range(5)      range(1, 5)      range(1, 5, 2)
[0,1,2,3,4] | [1,2,3,4] | [1,3]
```

## array(x1, ...), operator [x1, ...]

Creates an array from the function arguments.

The arguments must be constants and have types that have the smallest common type. At least one argument must be passed, because otherwise it isn't clear which type of array to create. That is, you can't use this function to create an empty array (to do that, use the 'emptyArray\*' function described above). Returns an 'Array(T)' type result, where 'T' is the smallest common type out of the passed arguments.

# arrayConcat

Combines arrays passed as arguments.

```
arrayConcat(arrays)
```

## Arguments

- arrays – Arbitrary number of arguments of **Array** type.

### Example

```
SELECT arrayConcat([1, 2], [3, 4], [5, 6]) AS res
```

```
res  
[1,2,3,4,5,6] |
```

# arrayElement(arr, n), operator arr[n]

Get the element with the index `n` from the array `arr`. `n` must be any integer type.

Indexes in an array begin from one.

Negative indexes are supported. In this case, it selects the corresponding element numbered from the end. For example, `arr[-1]` is the last item in the array.

If the index falls outside of the bounds of an array, it returns some default value (0 for numbers, an empty string for strings, etc.), except for the case with a non-constant array and a constant index 0 (in this case there will be an error `Array indices are 1-based`).

# has(arr, elem)

Checks whether the ‘arr’ array has the ‘elem’ element.

Returns 0 if the element is not in the array, or 1 if it is.

`NULL` is processed as a value.

```
SELECT has([1, 2, NULL], NULL)
```

```
has([1, 2, NULL], NULL)  
1 |
```

# hasAll

Checks whether one array is a subset of another.

```
hasAll(set, subset)
```

## Arguments

- set – Array of any type with a set of elements.
- subset – Array of any type with elements that should be tested to be a subset of `set`.

## Return values

- 1, if set contains all of the elements from subset.
- 0, otherwise.

### Peculiar properties

- An empty array is a subset of any array.
- Null processed as a value.
- Order of values in both of arrays does not matter.

### Examples

SELECT hasAll([], []) returns 1.

SELECT hasAll([1, Null], [Null]) returns 1.

SELECT hasAll([1.0, 2, 3, 4], [1, 3]) returns 1.

SELECT hasAll(['a', 'b'], ['a']) returns 1.

SELECT hasAll([1], ['a']) returns 0.

SELECT hasAll([[1, 2], [3, 4]], [[1, 2], [3, 5]]) returns 0.

## hasAny

Checks whether two arrays have intersection by some elements.

```
hasAny(array1, array2)
```

### Arguments

- array1 – Array of any type with a set of elements.
- array2 – Array of any type with a set of elements.

### Return values

- 1, if array1 and array2 have one similar element at least.
- 0, otherwise.

### Peculiar properties

- Null processed as a value.
- Order of values in both of arrays does not matter.

### Examples

SELECT hasAny([1], []) returns 0.

SELECT hasAny([Null], [Null, 1]) returns 1.

SELECT hasAny([-128, 1., 512], [1]) returns 1.

SELECT hasAny([[1, 2], [3, 4]], ['a', 'c']) returns 0.

SELECT hasAll([[1, 2], [3, 4]], [[1, 2], [1, 2]]) returns 1.

## hasSubstr

Checks whether all the elements of array2 appear in array1 in the same exact order. Therefore, the function will return 1, if and only if `array1 = prefix + array2 + suffix`.

```
hasSubstr(array1, array2)
```

In other words, the functions will check whether all the elements of array2 are contained in array1 like the `hasAll` function. In addition, it will check that the elements are observed in the same order in both array1 and array2.

For Example:

- `hasSubstr([1,2,3,4], [2,3])` returns 1. However, `hasSubstr([1,2,3,4], [3,2])` will return 0.
- `hasSubstr([1,2,3,4], [1,2,3])` returns 1. However, `hasSubstr([1,2,3,4], [1,2,4])` will return 0.

## Arguments

- `array1` – Array of any type with a set of elements.
- `array2` – Array of any type with a set of elements.

## Return values

- 1, if array1 contains array2.
- 0, otherwise.

## Peculiar properties

- The function will return 1 if array2 is empty.
- Null processed as a value. In other words `hasSubstr([1, 2, NULL, 3, 4], [2,3])` will return 0. However, `hasSubstr([1, 2, NULL, 3, 4], [2,NULL,3])` will return 1
- Order of values in both of arrays does matter.

## Examples

`SELECT hasSubstr([], [])` returns 1.

`SELECT hasSubstr([1, Null], [Null])` returns 1.

`SELECT hasSubstr([1.0, 2, 3, 4], [1, 3])` returns 0.

`SELECT hasSubstr(['a', 'b'], ['a'])` returns 1.

`SELECT hasSubstr(['a', 'b' , 'c'], ['a', 'b'])` returns 1.

`SELECT hasSubstr(['a', 'b' , 'c'], ['a', 'c'])` returns 0.

`SELECT hasSubstr([[1, 2], [3, 4], [5, 6]], [[1, 2], [3, 4]])` returns 1.

## indexOf(arr, x)

Returns the index of the first 'x' element (starting from 1) if it is in the array, or 0 if it is not.

Example:

```
SELECT indexOf([1, 3, NULL, NULL], NULL)
```

```
indexOf([1, 3, NULL, NULL], NULL)─  
      3 |
```

Elements set to `NULL` are handled as normal values.

## arrayCount([func,] arr1, ...)

Returns the number of elements in the arr array for which func returns something other than 0. If 'func' is not specified, it returns the number of non-zero elements in the array.

Note that the `arrayCount` is a [higher-order function](#). You can pass a lambda function to it as the first argument.

## countEqual(arr, x)

Returns the number of elements in the array equal to x. Equivalent to `arrayCount (elem -> elem = x, arr)`.

`NULL` elements are handled as separate values.

Example:

```
SELECT countEqual([1, 2, NULL, NULL], NULL)
```

```
countEqual([1, 2, NULL, NULL], NULL)─  
      2 |
```

## arrayEnumerate(arr)

Returns the array [1, 2, 3, ..., length (arr) ]

This function is normally used with ARRAY JOIN. It allows counting something just once for each array after applying ARRAY JOIN. Example:

```
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 |
```

In this example, `Reaches` is the number of conversions (the strings received after applying ARRAY JOIN), and `Hits` is the number of pageviews (strings before ARRAY JOIN). In this particular case, you can get the same result in an easier way:

```

SELECT
    sum(length(GoalsReached)) AS Reaches,
    count() AS Hits
FROM test.hits
WHERE (CounterID = 160656) AND notEmpty(GoalsReached)

```

Reaches	Hits
95606	31406

This function can also be used in higher-order functions. For example, you can use it to get array indexes for elements that match a condition.

## arrayEnumerateUniq(arr, ...)

Returns an array the same size as the source array, indicating for each element what its position is among elements with the same value.

For example: `arrayEnumerateUniq([10, 20, 10, 30]) = [1, 1, 2, 1]`.

This function is useful when using ARRAY JOIN and aggregation of array elements.

Example:

```

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

In this example, each goal ID has a calculation of the number of conversions (each element in the Goals nested data structure is a goal that was reached, which we refer to as a conversion) and the number of sessions. Without ARRAY JOIN, we would have counted the number of sessions as `sum(Sign)`. But in this particular case, the rows were multiplied by the nested Goals structure, so in order to count each session one time after this, we apply a condition to the value of the `arrayEnumerateUniq(Goals.ID)` function.

The `arrayEnumerateUniq` function can take multiple arrays of the same size as arguments. In this case, uniqueness is considered for tuples of elements in the same positions in all the arrays.

```

SELECT arrayEnumerateUniq([1, 1, 1, 2, 2, 2], [1, 1, 2, 1, 1, 2]) AS res

```

```
res
[1,2,1,1,2,1] |
```

This is necessary when using ARRAY JOIN with a nested data structure and further aggregation across multiple elements in this structure.

## arrayPopBack

Removes the last item from the array.

```
arrayPopBack(array)
```

### Arguments

- `array` – Array.

### Example

```
SELECT arrayPopBack([1, 2, 3]) AS res;
```

```
res
[1,2] |
```

## arrayPopFront

Removes the first item from the array.

```
arrayPopFront(array)
```

### Arguments

- `array` – Array.

### Example

```
SELECT arrayPopFront([1, 2, 3]) AS res;
```

```
res
[2,3] |
```

## arrayPushBack

Adds one item to the end of the array.

```
arrayPushBack(array, single_value)
```

### Arguments

- `array` – Array.

- `single_value` – A single value. Only numbers can be added to an array with numbers, and only strings can be added to an array of strings. When adding numbers, ClickHouse automatically sets the `single_value` type for the data type of the array. For more information about the types of data in ClickHouse, see “[Data types](#)”. Can be `NULL`. The function adds a `NULL` element to an array, and the type of array elements converts to `Nullable`.

## Example

```
SELECT arrayPushBack(['a'], 'b') AS res;
```

```
res  
['a','b'] |
```

## arrayPushFront

Adds one element to the beginning of the array.

```
arrayPushFront(array, single_value)
```

### Arguments

- `array` – Array.
- `single_value` – A single value. Only numbers can be added to an array with numbers, and only strings can be added to an array of strings. When adding numbers, ClickHouse automatically sets the `single_value` type for the data type of the array. For more information about the types of data in ClickHouse, see “[Data types](#)”. Can be `NULL`. The function adds a `NULL` element to an array, and the type of array elements converts to `Nullable`.

## Example

```
SELECT arrayPushFront(['b'], 'a') AS res;
```

```
res  
['a','b'] |
```

## arrayResize

Changes the length of the array.

```
arrayResize(array, size[, extender])
```

### Arguments:

- `array` — Array.
- `size` — Required length of the array.
  - If `size` is less than the original size of the array, the array is truncated from the right.
- If `size` is larger than the initial size of the array, the array is extended to the right with `extender` values or default values for the data type of the array items.
- `extender` — Value for extending an array. Can be `NULL`.

## Returned value:

An array of length `size`.

## Examples of calls

```
SELECT arrayResize([1], 3);
```

```
arrayResize([1], 3)─  
[1,0,0] |
```

```
SELECT arrayResize([1], 3, NULL);
```

```
arrayResize([1], 3, NULL)─  
[1,NULL,NULL] |
```

## arraySlice

Returns a slice of the array.

```
arraySlice(array, offset[, length])
```

### Arguments

- `array` – Array of data.
- `offset` – Indent from the edge of the array. A positive value indicates an offset on the left, and a negative value is an indent on the right. Numbering of the array items begins with 1.
- `length` – The length of the required slice. If you specify a negative value, the function returns an open slice `[offset, array_length - length]`. If you omit the value, the function returns the slice `[offset, the_end_of_array]`.

### Example

```
SELECT arraySlice([1, 2, NULL, 4, 5], 2, 3) AS res;
```

```
res─  
[2,NULL,4] |
```

Array elements set to `NULL` are handled as normal values.

## arraySort([func,] arr, ...)

Sorts the elements of the `arr` array in ascending order. If the `func` function is specified, sorting order is determined by the result of the `func` function applied to the elements of the array. If `func` accepts multiple arguments, the `arraySort` function is passed several arrays that the arguments of `func` will correspond to. Detailed examples are shown at the end of `arraySort` description.

Example of integer values sorting:

```
SELECT arraySort([1, 3, 3, 0]);
```

```
arraySort([1, 3, 3, 0])—  
[0,1,3,3] |
```

Example of string values sorting:

```
SELECT arraySort(['hello', 'world', '!']);
```

```
arraySort(['hello', 'world', '!'])—  
['!', 'hello', 'world'] |
```

Consider the following sorting order for the `NULL`, `NaN` and `Inf` values:

```
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` values are first in the array.
- `NULL` values are last in the array.
- `NaN` values are right before `NULL`.
- `Inf` values are right before `NaN`.

Note that `arraySort` is a **higher-order function**. You can pass a lambda function to it as the first argument. In this case, sorting order is determined by the result of the lambda function applied to the elements of the array.

Let's consider the following example:

```
SELECT arraySort((x) -> -x, [1, 2, 3]) as res;
```

```
res—  
[3,2,1] |
```

For each element of the source array, the lambda function returns the sorting key, that is,  $[1 \rightarrow -1, 2 \rightarrow -2, 3 \rightarrow -3]$ . Since the `arraySort` function sorts the keys in ascending order, the result is  $[3, 2, 1]$ . Thus, the `(x) -> -x` lambda function sets the **descending order** in a sorting.

The lambda function can accept multiple arguments. In this case, you need to pass the `arraySort` function several arrays of identical length that the arguments of lambda function will correspond to. The resulting array will consist of elements from the first input array; elements from the next input array(s) specify the sorting keys. For example:

```
SELECT arraySort((x, y) -> y, ['hello', 'world'], [2, 1]) as res;
```

```
res
['world', 'hello'] |
```

Here, the elements that are passed in the second array ([2, 1]) define a sorting key for the corresponding element from the source array (['hello', 'world']), that is, ['hello' -> 2, 'world' -> 1]. Since the lambda function does not use `x`, actual values of the source array do not affect the order in the result. So, 'hello' will be the second element in the result, and 'world' will be the first.

Other examples are shown below.

```
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] |
```

## Note

To improve sorting efficiency, the **Schwartzian transform** is used.

## arrayReverseSort([func,] arr, ...)

Sorts the elements of the `arr` array in descending order. If the `func` function is specified, `arr` is sorted according to the result of the `func` function applied to the elements of the array, and then the sorted array is reversed. If `func` accepts multiple arguments, the `arrayReverseSort` function is passed several arrays that the arguments of `func` will correspond to. Detailed examples are shown at the end of `arrayReverseSort` description.

Example of integer values sorting:

```
SELECT arrayReverseSort([1, 3, 3, 0]);
```

```
arrayReverseSort([1, 3, 3, 0])-
[3,3,1,0] |
```

Example of string values sorting:

```
SELECT arrayReverseSort(['hello', 'world', '!']);
```

```
arrayReverseSort(['hello', 'world', '!'])-
['world','hello','!'] |
```

Consider the following sorting order for the `NULL`, `NaN` and `Inf` values:

```
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` values are first in the array.
- `NULL` values are last in the array.
- `NaN` values are right before `NULL`.
- `-Inf` values are right before `NaN`.

Note that the `arrayReverseSort` is a [higher-order function](#). You can pass a lambda function to it as the first argument. Example is shown below.

```
SELECT arrayReverseSort((x) -> -x, [1, 2, 3]) as res;
```

```
res  
[1,2,3] |
```

The array is sorted in the following way:

1. At first, the source array (`[1, 2, 3]`) is sorted according to the result of the lambda function applied to the elements of the array. The result is an array `[3, 2, 1]`.
2. Array that is obtained on the previous step, is reversed. So, the final result is `[1, 2, 3]`.

The lambda function can accept multiple arguments. In this case, you need to pass the `arrayReverseSort` function several arrays of identical length that the arguments of lambda function will correspond to. The resulting array will consist of elements from the first input array; elements from the next input array(s) specify the sorting keys. For example:

```
SELECT arrayReverseSort((x, y) -> y, ['hello', 'world'], [2, 1]) as res;
```

```
res  
['hello','world'] |
```

In this example, the array is sorted in the following way:

1. At first, the source array (`['hello', 'world']`) is sorted according to the result of the lambda function applied to the elements of the arrays. The elements that are passed in the second array (`[2, 1]`), define the sorting keys for corresponding elements from the source array. The result is an array `['world', 'hello']`.
2. Array that was sorted on the previous step, is reversed. So, the final result is `['hello', 'world']`.

Other examples are shown below.

```
SELECT arrayReverseSort((x, y) -> y, [4, 3, 5], ['a', 'b', 'c']) AS res;
```

```
res
[5,3,4] |
```

```
SELECT arrayReverseSort((x, y) -> -y, [4, 3, 5], [1, 2, 3]) AS res;
```

```
res
[4,3,5] |
```

## arrayUniq(arr, ...)

If one argument is passed, it counts the number of different elements in the array.

If multiple arguments are passed, it counts the number of different tuples of elements at corresponding positions in multiple arrays.

If you want to get a list of unique items in an array, you can use `arrayReduce('groupUniqArray', arr)`.

## arrayJoin(arr)

A special function. See the section [“ArrayJoin function”](#).

## arrayDifference

Calculates the difference between adjacent array elements. Returns an array where the first element will be 0, the second is the difference between `a[1] - a[0]`, etc. The type of elements in the resulting array is determined by the type inference rules for subtraction (e.g. `UInt8 - UInt8 = Int16`).

### Syntax

```
arrayDifference(array)
```

### Arguments

- `array` – [Array](#).

### Returned values

Returns an array of differences between adjacent elements.

Type: [UInt\\*](#), [Int\\*](#), [Float\\*](#).

### Example

Query:

```
SELECT arrayDifference([1, 2, 3, 4]);
```

Result:

```
arrayDifference([1, 2, 3, 4])-
[0,1,1,1] |
```

Example of the overflow due to result type Int64:

Query:

```
SELECT arrayDifference([0, 100000000000000000000000]);
```

Result:

```
arrayDifference([0, 100000000000000000000000])  
[0,-8446744073709551616]
```

## arrayDistinct

Takes an array, returns an array containing the distinct elements only.

### Syntax

```
arrayDistinct(array)
```

### Arguments

- **array** – **Array**.

### Returned values

Returns an array containing the distinct elements.

### Example

Query:

```
SELECT arrayDistinct([1, 2, 2, 3, 1]);
```

Result:

```
arrayDistinct([1, 2, 2, 3, 1])  
[1,2,3]
```

## arrayEnumerateDense(arr)

Returns an array of the same size as the source array, indicating where each element first appears in the source array.

### Example:

```
SELECT arrayEnumerateDense([10, 20, 10, 30])
```

```
arrayEnumerateDense([10, 20, 10, 30])  
[1,2,1,3]
```

## arrayIntersect(arr)

Takes multiple arrays, returns an array with elements that are present in all source arrays.

Example:

```
SELECT
    arrayIntersect([1, 2], [1, 3], [2, 3]) AS no_intersect,
    arrayIntersect([1, 2], [1, 3], [1, 4]) AS intersect
```

```
graph LR; no_intersect[] --- intersect[1]
```

## arrayReduce

Applies an aggregate function to array elements and returns its result. The name of the aggregation function is passed as a string in single quotes 'max', 'sum'. When using parametric aggregate functions, the parameter is indicated after the function name in parentheses 'uniqUpTo(6)'.

### Syntax

```
arrayReduce(agg_func, arr1, arr2, ..., arrN)
```

### Arguments

- `agg_func` — The name of an aggregate function which should be a constant **string**.
- `arr` — Any number of **array** type columns as the parameters of the aggregation function.

### Returned value

### Example

Query:

```
SELECT arrayReduce('max', [1, 2, 3]);
```

Result:

```
graph LR; arrayReduce["arrayReduce('max', [1, 2, 3])"] --- result[3]
```

If an aggregate function takes multiple arguments, then this function must be applied to multiple arrays of the same size.

Query:

```
SELECT arrayReduce('maxIf', [3, 5], [1, 0]);
```

Result:

```
graph LR; arrayReduce["arrayReduce('maxIf', [3, 5], [1, 0])"] --- result[3]
```

Example with a parametric aggregate function:

Query:

```
SELECT arrayReduce('uniqUpTo(3)', [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]);
```

Result:

```
arrayReduce('uniqUpTo(3)', [1, 2, 3, 4, 5, 6, 7, 8, 9, 10])—  
4 |
```

## arrayReduceInRanges

Applies an aggregate function to array elements in given ranges and returns an array containing the result corresponding to each range. The function will return the same result as multiple `arrayReduce(agg_func, arraySlice(arr1, index, length), ...)`.

### Syntax

```
arrayReduceInRanges(agg_func, ranges, arr1, arr2, ..., arrN)
```

### Arguments

- `agg_func` — The name of an aggregate function which should be a constant **string**.
- `ranges` — The ranges to aggregate which should be an **array of tuples** which containing the index and the length of each range.
- `arr` — Any number of **Array** type columns as the parameters of the aggregation function.

### Returned value

- Array containing results of the aggregate function over specified ranges.

Type: **Array**.

### Example

Query:

```
SELECT arrayReduceInRanges(  
    'sum',  
    [(1, 5), (2, 3), (3, 4), (4, 4)],  
    [1000000, 200000, 30000, 4000, 500, 60, 7]  
) AS res
```

Result:

```
res—  
[1234500,234000,34560,4567] |
```

## arrayReverse(arr)

Returns an array of the same size as the original array containing the elements in reverse order.

Example:

```
SELECT arrayReverse([1, 2, 3])
```

```
arrayReverse([1, 2, 3])—  
[3,2,1]
```

## reverse(arr)

Synonym for “[arrayReverse](#)”

## arrayFlatten

Converts an array of arrays to a flat array.

Function:

- Applies to any depth of nested arrays.
- Does not change arrays that are already flat.

The flattened array contains all the elements from all source arrays.

### Syntax

```
flatten(array_of_arrays)
```

Alias: `flatten`.

### Arguments

- `array_of_arrays` — [Array](#) of arrays. For example, `[[1,2,3], [4,5]]`.

### Examples

```
SELECT flatten([[1]], [[2], [3]]);
```

```
flatten(array(array([1]), array([2], [3])))—  
[1,2,3]
```

## arrayCompact

Removes consecutive duplicate elements from an array. The order of result values is determined by the order in the source array.

### Syntax

```
arrayCompact(arr)
```

### Arguments

`arr` — The [array](#) to inspect.

### Returned value

The array without duplicate.

Type: `Array`.

## Example

Query:

```
SELECT arrayCompact([1, 1, nan, nan, 2, 3, 3, 3]);
```

Result:

```
arrayCompact([1, 1, nan, nan, 2, 3, 3, 3])—  
[1, nan, nan, 2, 3]
```

## arrayZip

Combines multiple arrays into a single array. The resulting array contains the corresponding elements of the source arrays grouped into tuples in the listed order of arguments.

### Syntax

```
arrayZip(arr1, arr2, ..., arrN)
```

### Arguments

- arrN — **Array**.

The function can take any number of arrays of different types. All the input arrays must be of equal size.

### Returned value

- Array with elements from the source arrays grouped into **tuples**. Data types in the tuple are the same as types of the input arrays and in the same order as arrays are passed.

Type: **Array**.

## Example

Query:

```
SELECT arrayZip(['a', 'b', 'c'], [5, 2, 1]);
```

Result:

```
arrayZip(['a', 'b', 'c'], [5, 2, 1])—  
[('a', 5), ('b', 2), ('c', 1)]
```

## arrayAUC

Calculate AUC (Area Under the Curve, which is a concept in machine learning, see more details: [https://en.wikipedia.org/wiki/Receiver\\_operating\\_characteristic#Area\\_under\\_the\\_curve](https://en.wikipedia.org/wiki/Receiver_operating_characteristic#Area_under_the_curve)).

### Syntax

```
arrayAUC(arr_scores, arr_labels)
```

### Arguments

- `arr_scores` — scores prediction model gives.
- `arr_labels` — labels of samples, usually 1 for positive sample and 0 for negative sample.

## Returned value

Returns AUC value with type Float64.

## Example

Query:

```
select arrayAUC([0.1, 0.4, 0.35, 0.8], [0, 0, 1, 1]);
```

Result:

```
arrayAUC([0.1, 0.4, 0.35, 0.8], [0, 0, 1, 1])—  
0.75 |
```

## arrayMap(func, arr1, ...)

Returns an array obtained from the original application of the `func` function to each element in the `arr` array.

Examples:

```
SELECT arrayMap(x -> (x + 2), [1, 2, 3]) as res;
```

```
res—  
[3,4,5] |
```

The following example shows how to create a tuple of elements from different arrays:

```
SELECT arrayMap((x, y) -> (x, y), [1, 2, 3], [4, 5, 6]) AS res
```

```
res—  
[(1,4),(2,5),(3,6)] |
```

Note that the `arrayMap` is a [higher-order function](#). You must pass a lambda function to it as the first argument, and it can't be omitted.

## arrayFilter(func, arr1, ...)

Returns an array containing only the elements in `arr1` for which `func` returns something other than 0.

Examples:

```
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] |
```

Note that the `arrayFilter` is a [higher-order function](#). You must pass a lambda function to it as the first argument, and it can't be omitted.

## arrayFill(func, arr1, ...)

Scan through `arr1` from the first element to the last element and replace `arr1[i]` by `arr1[i - 1]` if `func` returns 0. The first element of `arr1` will not be replaced.

Examples:

```
SELECT arrayFill(x -> notisNull(x), [1, null, 3, 11, 12, null, null, 5, 6, 14, null, null]) AS res
```

```
res
[1,1,3,11,12,12,12,5,6,14,14,14] |
```

Note that the `arrayFill` is a [higher-order function](#). You must pass a lambda function to it as the first argument, and it can't be omitted.

## arrayReverseFill(func, arr1, ...)

Scan through `arr1` from the last element to the first element and replace `arr1[i]` by `arr1[i + 1]` if `func` returns 0. The last element of `arr1` will not be replaced.

Examples:

```
SELECT arrayReverseFill(x -> notisNull(x), [1, null, 3, 11, 12, null, null, 5, 6, 14, null, null]) AS res
```

```
res
[1,3,3,11,12,5,5,5,6,14,NULL,NULL] |
```

Note that the `arrayReverseFill` is a [higher-order function](#). You must pass a lambda function to it as the first argument, and it can't be omitted.

## arraySplit(func, arr1, ...)

Split `arr1` into multiple arrays. When `func` returns something other than 0, the array will be split on the left hand side of the element. The array will not be split before the first element.

Examples:

```
SELECT arraySplit((x, y) -> y, [1, 2, 3, 4, 5], [1, 0, 0, 1, 0]) AS res
```

```
res  
[[1,2,3],[4,5]] |
```

Note that the `arraySplit` is a [higher-order function](#). You must pass a lambda function to it as the first argument, and it can't be omitted.

## arrayReverseSplit(func, arr1, ...)

Split `arr1` into multiple arrays. When `func` returns something other than 0, the array will be split on the right hand side of the element. The array will not be split after the last element.

Examples:

```
SELECT arrayReverseSplit((x, y) -> y, [1, 2, 3, 4, 5], [1, 0, 0, 1, 0]) AS res
```

```
res  
[[1],[2,3,4],[5]] |
```

Note that the `arrayReverseSplit` is a [higher-order function](#). You must pass a lambda function to it as the first argument, and it can't be omitted.

## arrayExists([func,] arr1, ...)

Returns 1 if there is at least one element in `arr` for which `func` returns something other than 0. Otherwise, it returns 0.

Note that the `arrayExists` is a [higher-order function](#). You can pass a lambda function to it as the first argument.

## arrayAll([func,] arr1, ...)

Returns 1 if `func` returns something other than 0 for all the elements in `arr`. Otherwise, it returns 0.

Note that the `arrayAll` is a [higher-order function](#). You can pass a lambda function to it as the first argument.

## arrayFirst(func, arr1, ...)

Returns the first element in the `arr1` array for which `func` returns something other than 0.

Note that the `arrayFirst` is a [higher-order function](#). You must pass a lambda function to it as the first argument, and it can't be omitted.

## arrayFirstIndex(func, arr1, ...)

Returns the index of the first element in the `arr1` array for which `func` returns something other than 0.

Note that the `arrayFirstIndex` is a [higher-order function](#). You must pass a lambda function to it as the first argument, and it can't be omitted.

## arrayMin

Returns the minimum of elements in the source array.

If the `func` function is specified, returns the mininum of elements converted by this function.

Note that the `arrayMin` is a [higher-order function](#). You can pass a lambda function to it as the first argument.

## Syntax

```
arrayMin([func,] arr)
```

## Arguments

- `func` — Function. [Expression](#).
- `arr` — Array. [Array](#).

## Returned value

- The minimum of function values (or the array minimum).

Type: if `func` is specified, matches `func` return value type, else matches the array elements type.

## Examples

Query:

```
SELECT arrayMin([1, 2, 4]) AS res;
```

Result:

```
res  
1 |
```

Query:

```
SELECT arrayMin(x -> (-x), [1, 2, 4]) AS res;
```

Result:

```
res  
-4 |
```

## arrayMax

Returns the maximum of elements in the source array.

If the `func` function is specified, returns the maximum of elements converted by this function.

Note that the `arrayMax` is a [higher-order function](#). You can pass a lambda function to it as the first argument.

## Syntax

```
arrayMax([func,] arr)
```

## Arguments

- `func` — Function. [Expression](#).
- `arr` — Array. [Array](#).

## Returned value

- The maximum of function values (or the array maximum).

Type: if `func` is specified, matches `func` return value type, else matches the array elements type.

## Examples

Query:

```
SELECT arrayMax([1, 2, 4]) AS res;
```

Result:

```
res
4 |
```

Query:

```
SELECT arrayMax(x -> (-x), [1, 2, 4]) AS res;
```

Result:

```
res
-1 |
```

# arraySum

Returns the sum of elements in the source array.

If the `func` function is specified, returns the sum of elements converted by this function.

Note that the `arraySum` is a [higher-order function](#). You can pass a lambda function to it as the first argument.

## Syntax

```
arraySum([func,] arr)
```

## Arguments

- `func` — Function. [Expression](#).
- `arr` — Array. [Array](#).

## Returned value

- The sum of the function values (or the array sum).

Type: for decimal numbers in source array (or for converted values, if `func` is specified) — [Decimal128](#), for floating point numbers — [Float64](#), for numeric unsigned — [UInt64](#), and for numeric signed — [Int64](#).

## Examples

Query:

```
SELECT arraySum([2, 3]) AS res;
```

Result:

```
res  
5 |
```

Query:

```
SELECT arraySum(x -> x*x, [2, 3]) AS res;
```

Result:

```
res  
13 |
```

## arrayAvg

Returns the average of elements in the source array.

If the `func` function is specified, returns the average of elements converted by this function.

Note that the `arrayAvg` is a **higher-order function**. You can pass a lambda function to it as the first argument.

### Syntax

```
arrayAvg([func,] arr)
```

### Arguments

- `func` — Function. [Expression](#).
- `arr` — Array. [Array](#).

### Returned value

- The average of function values (or the array average).

Type: [Float64](#).

### Examples

Query:

```
SELECT arrayAvg([1, 2, 4]) AS res;
```

Result:

```
res  
2.3333333333333335 |
```

Query:

```
SELECT arrayAvg(x -> (x * x), [2, 4]) AS res;
```

Result:

```
res  
10 |
```

## arrayCumSum([func,] arr1, ...)

Returns an array of partial sums of elements in the source array (a running sum). If the `func` function is specified, then the values of the array elements are converted by this function before summing.

Example:

```
SELECT arrayCumSum([1, 1, 1, 1]) AS res
```

```
res  
[1, 2, 3, 4] |
```

Note that the `arrayCumSum` is a [higher-order function](#). You can pass a lambda function to it as the first argument.

## arrayCumSumNonNegative(arr)

Same as `arrayCumSum`, returns an array of partial sums of elements in the source array (a running sum). Different `arrayCumSum`, when then returned value contains a value less than zero, the value is replace with zero and the subsequent calculation is performed with zero parameters. For example:

```
SELECT arrayCumSumNonNegative([1, 1, -4, 1]) AS res
```

```
res  
[1, 2, 0, 1] |
```

Note that the `arraySumNonNegative` is a [higher-order function](#). You can pass a lambda function to it as the first argument.

## arrayProduct

Multiplies elements of an [array](#).

### Syntax

```
arrayProduct(arr)
```

### Arguments

- `arr` — [Array](#) of numeric values.

### Returned value

- A product of array's elements.

Type: **Float64**.

## Examples

Query:

```
SELECT arrayProduct([1,2,3,4,5,6]) as res;
```

Result:

res
720

Query:

```
SELECT arrayProduct([toDecimal64(1,8), toDecimal64(2,8), toDecimal64(3,8)]) as res, toTypeName(res);
```

Return value type is always **Float64**. Result:

res
6

toTypeName(arrayProduct(array(toDecimal64(1, 8), toDecimal64(2, 8), toDecimal64(3, 8))))

## Comparison Functions

Comparison functions always return 0 or 1 (Uint8).

The following types can be compared:

- numbers
- strings and fixed strings
- dates
- dates with times

within each group, but not between different groups.

For example, you can't compare a date with a string. You have to use a function to convert the string to a date, or vice versa.

Strings are compared by bytes. A shorter string is smaller than all strings that start with it and that contain at least one more character.

**equals**, **a = b** and **a == b** operator

**notEquals**, **a != b** and **a \<> b** operator

**less**, **\<** operator

**greater**, **>** operator

**lessOrEquals**, **\<=** operator

## greaterOrEquals, $\geq$ operator

# Logical Functions

Performs logical operations on arguments of any numeric types, but returns a **UInt8** number equal to 0, 1 or **NULL** in some cases.

Zero as an argument is considered **false**, while any non-zero value is considered **true**.

## and

Calculates the result of the logical conjunction between two or more values. Corresponds to [Logical AND Operator](#).

### Syntax

```
and(val1, val2...)
```

You can use the [short\\_circuit\\_function\\_evaluation](#) setting to calculate the **and** function according to a short scheme. If this setting is enabled, **val<sub>i</sub>** is evaluated only on rows where **(val<sub>1</sub> AND val<sub>2</sub> AND ... AND val<sub>{i-1}</sub>)** is true. For example, an exception about division by zero is not thrown when executing the query **SELECT and(number = 2, intDiv(1, number)) FROM numbers(10)**.

### Arguments

- **val<sub>1</sub>, val<sub>2</sub>, ...** — List of at least two values. **Int**, **UInt**, **Float** or **Nullable**.

### Returned value

- **0**, if there is at least one zero value argument.
- **NULL**, if there are no zero values arguments and there is at least one **NULL** argument.
- **1**, otherwise.

Type: **UInt8** or **Nullable(UInt8)**.

### Example

Query:

```
SELECT and(0, 1, -2);
```

Result:

```
└─and(0, 1, -2)─  
   0 |
```

With **NULL**:

```
SELECT and(NULL, 1, 10, -2);
```

Result:

```
and(NULL, 1, 10, -2)
    NULL |

```

## Or

Calculates the result of the logical disjunction between two or more values. Corresponds to [Logical OR Operator](#).

### Syntax

```
or(val1, val2...)
```

You can use the [short\\_circuit\\_function\\_evaluation](#) setting to calculate the `or` function according to a short scheme. If this setting is enabled, `vali` is evaluated only on rows where `((NOT val1) AND (NOT val2) AND ... AND (NOT val{i-1}))` is true. For example, an exception about division by zero is not thrown when executing the query `SELECT or(number = 0, intDiv(1, number) != 0) FROM numbers(10)`

### Arguments

- `val1, val2, ...` — List of at least two values. [Int](#), [UInt](#), [Float](#) or [Nullable](#).

### Returned value

- `1`, if there is at least one non-zero value.
- `0`, if there are only zero values.
- `NULL`, if there are only zero values and `NULL`.

Type: [UInt8](#) or [Nullable\(UInt8\)](#).

### Example

Query:

```
SELECT or(1, 0, 0, 2, NULL);
```

Result:

```
or(1, 0, 0, 2, NULL)
    1 |

```

With `NULL`:

```
SELECT or(0, NULL);
```

Result:

```
or(0, NULL)
    NULL |

```

not

Calculates the result of the logical negation of the value. Corresponds to [Logical Negation Operator](#).

## Syntax

```
not(val);
```

## Arguments

- `val` — The value. [Int](#), [UInt](#), [Float](#) or [Nullable](#).

## Returned value

- `1`, if the `val` is `0`.
- `0`, if the `val` is a non-zero value.
- `NULL`, if the `val` is a `NULL` value.

Type: [UInt8](#) or [Nullable\(UInt8\)](#).

## Example

Query:

```
SELECT NOT(1);
```

Result:

not(1)
0

## XOR

Calculates the result of the logical exclusive disjunction between two or more values. For more than two values the function works as if it calculates `XOR` of the first two values and then uses the result with the next value to calculate `XOR` and so on.

## Syntax

```
xor(val1, val2...)
```

## Arguments

- `val1, val2, ...` — List of at least two values. [Int](#), [UInt](#), [Float](#) or [Nullable](#).

## Returned value

- `1`, for two values: if one of the values is zero and other is not.
- `0`, for two values: if both values are zero or non-zero at the same time.
- `NULL`, if there is at least one `NULL` value.

Type: [UInt8](#) or [Nullable\(UInt8\)](#).

## Example

Query:

```
SELECT xor(0, 1, 1);
```

Result:

```
└─xor(0, 1, 1)─  
   0 |
```

## Type Conversion Functions

### Common Issues of Numeric Conversions

When you convert a value from one to another data type, you should remember that in common case, it is an unsafe operation that can lead to a data loss. A data loss can occur if you try to fit value from a larger data type to a smaller data type, or if you convert values between different data types.

ClickHouse has the [same behavior as C++ programs](#).

## toInt(8|16|32|64|128|256)

Converts an input value to the [Int](#) data type. This function family includes:

- `toInt8(expr)` — Results in the `Int8` data type.
- `toInt16(expr)` — Results in the `Int16` data type.
- `toInt32(expr)` — Results in the `Int32` data type.
- `toInt64(expr)` — Results in the `Int64` data type.
- `toInt128(expr)` — Results in the `Int128` data type.
- `toInt256(expr)` — Results in the `Int256` data type.

#### Arguments

- `expr` — [Expression](#) returning a number or a string with the decimal representation of a number. Binary, octal, and hexadecimal representations of numbers are not supported. Leading zeroes are stripped.

#### Returned value

Integer value in the `Int8`, `Int16`, `Int32`, `Int64`, `Int128` or `Int256` data type.

Functions use [rounding towards zero](#), meaning they truncate fractional digits of numbers.

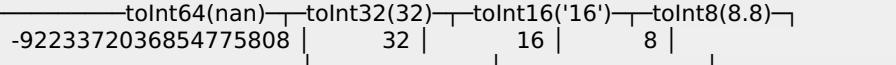
The behavior of functions for the [NaN](#) and [Inf](#) arguments is undefined. Remember about [numeric conversions issues](#), when using the functions.

#### Example

Query:

```
SELECT toInt64(nan), toInt32(32), toInt16('16'), toInt8(8.8);
```

Result:



## toInt(8|16|32|64|128|256)OrZero

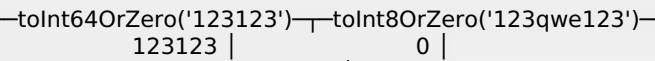
It takes an argument of type String and tries to parse it into Int (8 | 16 | 32 | 64 | 128 | 256). If failed, returns 0.

### Example

Query:

```
SELECT toInt64OrZero('123123'), toInt8OrZero('123qwe123');
```

Result:



## toInt(8|16|32|64|128|256)OrNull

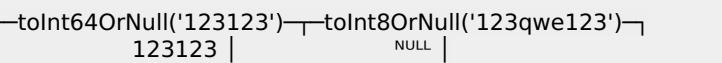
It takes an argument of type String and tries to parse it into Int (8 | 16 | 32 | 64 | 128 | 256). If failed, returns NULL.

### Example

Query:

```
SELECT toInt64OrNull('123123'), toInt8OrNull('123qwe123');
```

Result:



## toInt(8|16|32|64|128|256)OrDefault

It takes an argument of type String and tries to parse it into Int (8 | 16 | 32 | 64 | 128 | 256). If failed, returns the default type value.

### Example

Query:

```
SELECT toInt64OrDefault('123123', cast('-1' as Int64)), toInt8OrDefault('123qwe123', cast('-1' as Int8));
```

Result:

```
toInt64OrDefault('123123', CAST('-1', 'Int64'))-->toInt8OrDefault('123qwe123', CAST('-1', 'Int8'))-->
    123123 |           -1 |
```

## toUInt(8|16|32|64|256)

Converts an input value to the [UInt](#) data type. This function family includes:

- `toUInt8(expr)` — Results in the `UInt8` data type.
- `toUInt16(expr)` — Results in the `UInt16` data type.
- `toUInt32(expr)` — Results in the `UInt32` data type.
- `toUInt64(expr)` — Results in the `UInt64` data type.
- `toUInt256(expr)` — Results in the `UInt256` data type.

### Arguments

- `expr` — [Expression](#) returning a number or a string with the decimal representation of a number. Binary, octal, and hexadecimal representations of numbers are not supported. Leading zeroes are stripped.

### Returned value

Integer value in the `UInt8`, `UInt16`, `UInt32`, `UInt64` or `UInt256` data type.

Functions use [rounding towards zero](#), meaning they truncate fractional digits of numbers.

The behavior of functions for negative arguments and for the [NaN and Inf](#) arguments is undefined. If you pass a string with a negative number, for example `'-32'`, ClickHouse raises an exception. Remember about [numeric conversions issues](#), when using the functions.

### Example

Query:

```
SELECT toUInt64(nan), toUInt32(-32), toUInt16('16'), toUInt8(8.8);
```

Result:

```
toUInt64(nan)---toUInt32(-32)---toUInt16('16')---toUInt8(8.8)-->
    9223372036854775808 | 4294967264 | 16 | 8 |
```

## toUInt(8|16|32|64|256)OrZero

## toUInt(8|16|32|64|256)OrNull

## toUInt(8|16|32|64|256)OrDefault

## toFloat(32|64)

## toFloat(32|64)OrZero

## toFloat(32|64)OrNull

`toFloat(32|64)OrDefault`

`toDate`

Alias: `DATE`.

`toDateOrZero`

`toDateOrNull`

`toDateOrDefault`

`toDateTime`

`toDateTimeOrZero`

`toDateTimeOrNull`

`toDateTimeOrDefault`

`toDate32`

Converts the argument to the `Date32` data type. If the value is outside the range returns the border values supported by `Date32`. If the argument has `Date` type, borders of `Date` are taken into account.

## Syntax

```
toDate32(expr)
```

## Arguments

- `expr` — The value. `String`, `UInt32` or `Date`.

## Returned value

- A calendar date.

Type: `Date32`.

## Example

1. The value is within the range:

```
SELECT toDate32('1955-01-01') AS value, toTypeName(value);
```

value	toTypeName(toDate32('1925-01-01'))
1955-01-01	Date32

2. The value is outside the range:

```
SELECT toDate32('1924-01-01') AS value, toTypeName(value);
```

```
value---toTypeName(toDate32('1925-01-01'))---  
1925-01-01 | Date32 |
```

3. With `Date`-type argument:

```
SELECT toDate32(toDate('1924-01-01')) AS value, toTypeName(value);
```

```
value---toTypeName(toDate32(toDate('1924-01-01'))---  
1970-01-01 | Date32 |
```

## toDate32OrZero

The same as `toDate32` but returns the min value of `Date32` if invalid argument is received.

### Example

Query:

```
SELECT toDate32OrZero('1924-01-01'), toDate32OrZero('');
```

Result:

```
toDate32OrZero('1924-01-01')---toDate32OrZero('')---  
1925-01-01 | 1925-01-01 |
```

## toDate32OrNull

The same as `toDate32` but returns `NULL` if invalid argument is received.

### Example

Query:

```
SELECT toDate32OrNull('1955-01-01'), toDate32OrNull('');
```

Result:

```
toDate32OrNull('1955-01-01')---toDate32OrNull('')---  
1955-01-01 | NULL |
```

## toDate32OrDefault

Converts the argument to the `Date32` data type. If the value is outside the range returns the lower border value supported by `Date32`. If the argument has `Date` type, borders of `Date` are taken into account. Returns default value if invalid argument is received.

### Example

Query:

```
SELECT
    toDate32OrDefault('1930-01-01', toDate32('2020-01-01')),
    toDate32OrDefault('xx1930-01-01', toDate32('2020-01-01'));
```

Result:

toDate32OrDefault('1930-01-01', toDate32('2020-01-01'))	toDate32OrDefault('xx1930-01-01', toDate32('2020-01-01'))
1930-01-01	2020-01-01

## toDecimal(32|64|128|256)

Converts `value` to the **Decimal** data type with precision of `S`. The `value` can be a number or a string. The `S` (scale) parameter specifies the number of decimal places.

- `toDecimal32(value, S)`
- `toDecimal64(value, S)`
- `toDecimal128(value, S)`
- `toDecimal256(value, S)`

## toDecimal(32|64|128|256)OrNull

Converts an input string to a **Nullable(Decimal(P,S))** data type value. This family of functions include:

- `toDecimal32OrNull(expr, S)` — Results in `Nullable(Decimal32(S))` data type.
- `toDecimal64OrNull(expr, S)` — Results in `Nullable(Decimal64(S))` data type.
- `toDecimal128OrNull(expr, S)` — Results in `Nullable(Decimal128(S))` data type.
- `toDecimal256OrNull(expr, S)` — Results in `Nullable(Decimal256(S))` data type.

These functions should be used instead of `toDecimal*()` functions, if you prefer to get a `NUL` value instead of an exception in the event of an input value parsing error.

### Arguments

- `expr` — **Expression**, returns a value in the **String** data type. ClickHouse expects the textual representation of the decimal number. For example, '`1.111`'.
- `S` — Scale, the number of decimal places in the resulting value.

### Returned value

A value in the `Nullable(Decimal(P,S))` data type. The value contains:

- Number with `S` decimal places, if ClickHouse interprets the input string as a number.
- `NUL`, if ClickHouse can't interpret the input string as a number or if the input number contains more than `S` decimal places.

### Examples

Query:

```
SELECT toDecimal32OrNull(toString(-1.111), 5) AS val, toTypeName(val);
```

Result:

val	toTypeName(toDecimal32OrNull(toString(-1.111), 5))
-1.11100	Nullable(Decimal(9, 5))

Query:

```
SELECT toDecimal32OrNull(toString(-1.111), 2) AS val, toTypeName(val);
```

Result:

val	toTypeName(toDecimal32OrNull(toString(-1.111), 2))
NULL	Nullable(Decimal(9, 2))

## toDecimal(32|64|128|256)OrDefault

Converts an input string to a [Decimal\(P,S\)](#) data type value. This family of functions include:

- `toDecimal32OrDefault(expr, S)` — Results in `Decimal32(S)` data type.
- `toDecimal64OrDefault(expr, S)` — Results in `Decimal64(S)` data type.
- `toDecimal128OrDefault(expr, S)` — Results in `Decimal128(S)` data type.
- `toDecimal256OrDefault(expr, S)` — Results in `Decimal256(S)` data type.

These functions should be used instead of `toDecimal*`() functions, if you prefer to get a default value instead of an exception in the event of an input value parsing error.

### Arguments

- `expr` — [Expression](#), returns a value in the `String` data type. ClickHouse expects the textual representation of the decimal number. For example, '`1.111`'.
- `S` — Scale, the number of decimal places in the resulting value.

### Returned value

A value in the `Decimal(P,S)` data type. The value contains:

- Number with `S` decimal places, if ClickHouse interprets the input string as a number.
- Default `Decimal(P,S)` data type value, if ClickHouse can't interpret the input string as a number or if the input number contains more than `S` decimal places.

### Examples

Query:

```
SELECT toDecimal32OrDefault(toString(-1.111), 5) AS val, toTypeName(val);
```

Result:

```
val -> typeName(toDecimal32OrDefault(toString(-1.111), 5)) ->
-1.111 | Decimal(9, 5) |
```

Query:

```
SELECT toDecimal32OrDefault(toString(-1.111), 2) AS val, typeName(val);
```

Result:

```
val -> typeName(toDecimal32OrDefault(toString(-1.111), 2)) ->
0 | Decimal(9, 2) |
```

## toDecimal(32|64|128|256)OrZero

Converts an input value to the [Decimal\(P,S\)](#) data type. This family of functions include:

- `toDecimal32OrZero(expr, S)` — Results in `Decimal32(S)` data type.
- `toDecimal64OrZero(expr, S)` — Results in `Decimal64(S)` data type.
- `toDecimal128OrZero(expr, S)` — Results in `Decimal128(S)` data type.
- `toDecimal256OrZero(expr, S)` — Results in `Decimal256(S)` data type.

These functions should be used instead of `toDecimal(*)` functions, if you prefer to get a `0` value instead of an exception in the event of an input value parsing error.

### Arguments

- `expr` — [Expression](#), returns a value in the `String` data type. ClickHouse expects the textual representation of the decimal number. For example, '`1.111`'.
- `S` — Scale, the number of decimal places in the resulting value.

### Returned value

A value in the `Nullable(Decimal(P,S))` data type. The value contains:

- Number with `S` decimal places, if ClickHouse interprets the input string as a number.
- `0` with `S` decimal places, if ClickHouse can't interpret the input string as a number or if the input number contains more than `S` decimal places.

### Example

Query:

```
SELECT toDecimal32OrZero(toString(-1.111), 5) AS val, typeName(val);
```

Result:

```
val -> typeName(toDecimal32OrZero(toString(-1.111), 5)) ->
-1.11100 | Decimal(9, 5) |
```

Query:

```
SELECT toDecimal32OrZero(toString(-1.111), 2) AS val, toTypeName(val);
```

Result:

val	toTypeName(toDecimal32OrZero(toString(-1.111), 2))
0.00	Decimal(9, 2)

## toString

Functions for converting between numbers, strings (but not fixed strings), dates, and dates with times. All these functions accept one argument.

When converting to or from a string, the value is formatted or parsed using the same rules as for the TabSeparated format (and almost all other text formats). If the string can't be parsed, an exception is thrown and the request is canceled.

When converting dates to numbers or vice versa, the date corresponds to the number of days since the beginning of the Unix epoch.

When converting dates with times to numbers or vice versa, the date with time corresponds to the number of seconds since the beginning of the Unix epoch.

The date and date-with-time formats for the toDate/toDateTime functions are defined as follows:

YYYY-MM-DD
YYYY-MM-DD hh:mm:ss

As an exception, if converting from UInt32, Int32, UInt64, or Int64 numeric types to Date, and if the number is greater than or equal to 65536, the number is interpreted as a Unix timestamp (and not as the number of days) and is rounded to the date. This allows support for the common occurrence of writing 'toDate(unix\_timestamp)', which otherwise would be an error and would require writing the more cumbersome 'toDate(toDateTime(unix\_timestamp))'.

Conversion between a date and date with time is performed the natural way: by adding a null time or dropping the time.

Conversion between numeric types uses the same rules as assignments between different numeric types in C++.

Additionally, the toString function of the DateTime argument can take a second String argument containing the name of the time zone. Example: Asia/Yekaterinburg In this case, the time is formatted according to the specified time zone.

### Example

Query:

```
SELECT
    now() AS now_local,
    toString(now(), 'Asia/Yekaterinburg') AS now_yekat;
```

Result:

now_local	now_yekat
2016-06-15 00:11:21	2016-06-15 02:11:21

Also see the `toUnixTimestamp` function.

## `toFixedString(s, N)`

Converts a String type argument to a `FixedString(N)` type (a string with fixed length N). N must be a constant.

If the string has fewer bytes than N, it is padded with null bytes to the right. If the string has more bytes than N, an exception is thrown.

## `toStringCutToZero(s)`

Accepts a String or `FixedString` argument. Returns the String with the content truncated at the first zero byte found.

### **Example**

Query:

```
SELECT toFixedString('foo', 8) AS s, toStringCutToZero(s) AS s_cut;
```

Result:

s								s cut	
foo\0\0\0\0\0									

Query:

```
SELECT toFixedString('foo\0bar', 8) AS s, toStringCutToZero(s) AS s_cut;
```

Result:

s					s cut	
foo\0bar\0						

## `reinterpretAsUInt(8|16|32|64)`

## `reinterpretAsInt(8|16|32|64)`

## `reinterpretAsFloat(32|64)`

## `reinterpretAsDate`

## `reinterpretAsDateTime`

These functions accept a string and interpret the bytes placed at the beginning of the string as a number in host order (little endian). If the string isn't long enough, the functions work as if the string is padded with the necessary number of null bytes. If the string is longer than needed, the extra bytes are ignored. A date is interpreted as the number of days since the beginning of the Unix Epoch, and a date with time is interpreted as the number of seconds since the beginning of the Unix Epoch.

## `reinterpretAsString`

This function accepts a number or date or date with time, and returns a string containing bytes representing the corresponding value in host order (little endian). Null bytes are dropped from the end. For example, a UInt32 type value of 255 is a string that is one byte long.

## reinterpretAsFixedString

This function accepts a number or date or date with time, and returns a FixedString containing bytes representing the corresponding value in host order (little endian). Null bytes are dropped from the end. For example, a UInt32 type value of 255 is a FixedString that is one byte long.

## reinterpretAsUUID

Accepts 16 bytes string and returns UUID containing bytes representing the corresponding value in network byte order (big-endian). If the string isn't long enough, the function works as if the string is padded with the necessary number of null bytes to the end. If the string longer than 16 bytes, the extra bytes at the end are ignored.

### Syntax

```
reinterpretAsUUID(fixed_string)
```

### Arguments

- fixed\_string — Big-endian byte string. [FixedString](#).

### Returned value

- The UUID type value. [UUID](#).

### Examples

String to UUID.

Query:

```
SELECT reinterpretAsUUID(reverse(unhex('000102030405060708090a0b0c0d0e0f')));
```

Result:

```
reinterpretAsUUID(reverse(unhex('000102030405060708090a0b0c0d0e0f')))—  
08090a0b-0c0d-0e0f-0001-020304050607 |
```

Going back and forth from String to UUID.

Query:

```
WITH  
    generateUUIDv4() AS uuid,  
    identity(lower(hex(reverse(reinterpretAsString(uuid))))) AS str,  
    reinterpretAsUUID(reverse(unhex(str))) AS uuid2  
SELECT uuid = uuid2;
```

Result:

```
equals(uuid, uuid2)
  1 |
```

## reinterpret(x, T)

Uses the same source in-memory bytes sequence for `x` value and reinterprets it to destination type.

### Syntax

```
reinterpret(x, type)
```

### Arguments

- `x` — Any type.
- `type` — Destination type. [String](#).

### Returned value

- Destination type value.

### Examples

Query:

```
SELECT reinterpret(toInt8(-1), 'UInt8') as int_to_uint,
       reinterpret(toInt8(1), 'Float32') as int_to_float,
       reinterpret('1', 'UInt32') as string_to_int;
```

Result:

```
int_to_uint---int_to_float---string_to_int---|
  255 |      1e-45 |        49 |
```

## CAST(x, T)

Converts an input value to the specified data type. Unlike the [reinterpret](#) function, `CAST` tries to present the same value using the new data type. If the conversion can not be done then an exception is raised. Several syntax variants are supported.

### Syntax

```
CAST(x, T)
CAST(x AS t)
x::t
```

### Arguments

- `x` — A value to convert. May be of any type.
- `T` — The name of the target data type. [String](#).
- `t` — The target data type.

### Returned value

- Converted value.

## Note

If the input value does not fit the bounds of the target type, the result overflows. For example, `CAST(-1, 'UInt8')` returns 255.

## Examples

Query:

```
SELECT
    CAST(toInt8(-1), 'UInt8') AS cast_int_to_uint,
    CAST(1.5 AS Decimal(3,2)) AS cast_float_to_decimal,
    '1'::Int32 AS cast_string_to_int;
```

Result:

cast_int_to_uint	cast_float_to_decimal	cast_string_to_int
255	1.50	1

Query:

```
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;
```

Result:

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

Conversion to `FixedString(N)` only works for arguments of type `String` or `FixedString`.

Type conversion to `Nullable` and back is supported.

## Example

Query:

```
SELECT toTypeName(x) FROM t_null;
```

Result:

toTypeName(x)
Int8
Int8

Query:

```
SELECT toTypeName(CAST(x, 'Nullable(UInt16)')) FROM t_null;
```

Result:

```
toTypeName(CAST(x, 'Nullable(UInt16)'))  
 Nullable(UInt16)  
 Nullable(UInt16)
```

## See also

- [cast\\_keep\\_nullable](#) setting

## accurateCast(x, T)

Converts `x` to the `T` data type.

The difference from [cast\(x, T\)](#) is that `accurateCast` does not allow overflow of numeric types during cast if type value `x` does not fit the bounds of type `T`. For example, `accurateCast(-1, 'UInt8')` throws an exception.

## Example

Query:

```
SELECT cast(-1, 'UInt8') as uint8;
```

Result:

```
uint8  
 255 |
```

Query:

```
SELECT accurateCast(-1, 'UInt8') as uint8;
```

Result:

```
Code: 70. DB::Exception: Received from localhost:9000. DB::Exception: Value in column Int8 cannot be safely converted into type UInt8: While processing accurateCast(-1, 'UInt8') AS uint8.
```

## accurateCastOrNull(x, T)

Converts input value `x` to the specified data type `T`. Always returns `Nullable` type and returns `NULL` if the casted value is not representable in the target type.

## Syntax

```
accurateCastOrNull(x, T)
```

## Parameters

- `x` — Input value.

- `T` — The name of the returned data type.

## Returned value

- The value, converted to the specified data type `T`.

## Example

Query:

```
SELECT toTypeName(accurateCastOrNull(5, 'UInt8'));
```

Result:

```
toTypeName(accurateCastOrNull(5, 'UInt8'))  
|  
Nullable(UInt8)
```

Query:

```
SELECT  
    accurateCastOrNull(-1, 'UInt8') as uint8,  
    accurateCastOrNull(128, 'Int8') as int8,  
    accurateCastOrNull('Test', 'FixedString(2)') as fixed_string;
```

Result:

```
uint8 int8 fixed_string  
NULL NULL NULL
```

## accurateCastOrDefault(`x`, `T`[, `default_value`])

Converts input value `x` to the specified data type `T`. Returns default type value or `default_value` if specified if the casted value is not representable in the target type.

## Syntax

```
accurateCastOrDefault(x, T)
```

## Parameters

- `x` — Input value.
- `T` — The name of the returned data type.
- `default_value` — Default value of returned data type.

## Returned value

- The value converted to the specified data type `T`.

## Example

Query:

```
SELECT toTypeName(accurateCastOrDefault(5, 'UInt8'));
```

Result:

```
toTypeName(accurateCastOrDefault(5, 'UInt8'))—  
 UInt8
```

Query:

```
SELECT  
    accurateCastOrDefault(-1, 'UInt8') as uint8,  
    accurateCastOrDefault(-1, 'UInt8', 5) as uint8_default,  
    accurateCastOrDefault(128, 'Int8') as int8,  
    accurateCastOrDefault(128, 'Int8', 5) as int8_default,  
    accurateCastOrDefault('Test', 'FixedString(2)') as fixed_string,  
    accurateCastOrDefault('Test', 'FixedString(2)', 'Te') as fixed_string_default;
```

Result:

```
uint8—uint8_default—int8—int8_default—fixed_string—fixed_string_default—  
 0 |      5 |  0 |      5 |           | Te |
```

## toInterval(Year|Quarter|Month|Week|Day|Hour|Minute|Second)

Converts a Number type argument to an [Interval](#) data type.

### Syntax

```
toIntervalSecond(number)  
toIntervalMinute(number)  
toIntervalHour(number)  
toIntervalDay(number)  
toIntervalWeek(number)  
toIntervalMonth(number)  
toIntervalQuarter(number)  
toIntervalYear(number)
```

### Arguments

- number — Duration of interval. Positive integer number.

### Returned values

- The value in [Interval](#) data type.

### Example

Query:

```
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;
```

Result:

```
plus(date, interval_week)---plus(date, interval_to_week)---  
2019-01-08 | 2019-01-08 |
```

## parseDateTimeBestEffort

## parseDateTime32BestEffort

Converts a date and time in the [String](#) representation to [DateTime](#) data type.

The function parses [ISO 8601](#), [RFC 1123 - 5.2.14 RFC-822 Date and Time Specification](#), ClickHouse's and some other date and time formats.

### Syntax

```
parseDateTimeBestEffort(time_string [, time_zone])
```

### Arguments

- `time_string` — String containing a date and time to convert. [String](#).
- `time_zone` — Time zone. The function parses `time_string` according to the time zone. [String](#).

### Supported non-standard formats

- A string containing 9..10 digit [unix timestamp](#).
- A string with a date and a time component: `YYYYMMDDhhmmss`, `DD/MM/YYYY hh:mm:ss`, `DD-MM-YY hh:mm`, `YYYY-MM-DD hh:mm:ss`, etc.
- A string with a date, but no time component: `YYYY`, `YYYYMM`, `YYYY*MM`, `DD/MM/YYYY`, `DD-MM-YY` etc.
- A string with a day and time: `DD`, `DD hh`, `DD hh:mm`. In this case `YYYY-MM` are substituted as `2000-01`.
- A string that includes the date and time along with time zone offset information: `YYYY-MM-DD hh:mm:ss ±h:mm`, etc. For example, `2020-12-12 17:36:00 -5:00`.

For all of the formats with separator the function parses months names expressed by their full name or by the first three letters of a month name. Examples: `24/DEC/18`, `24-Dec-18`, `01-September-2018`.

### Returned value

- `time_string` converted to the [DateTime](#) data type.

### Examples

Query:

```
SELECT parseDateTimeBestEffort('12/12/2020 12:12:57')  
AS parseDateTimeBestEffort;
```

Result:

```
parseDateTimeBestEffort---  
2020-12-12 12:12:57 |
```

Query:

```
SELECT parseDateTimeBestEffort('Sat, 18 Aug 2018 07:22:16 GMT', 'Europe/Moscow')
AS parseDateTimeBestEffort;
```

Result:

```
parseDateTimeBestEffort
2018-08-18 10:22:16 |
```

Query:

```
SELECT parseDateTimeBestEffort('1284101485')
AS parseDateTimeBestEffort;
```

Result:

```
parseDateTimeBestEffort
2015-07-07 12:04:41 |
```

Query:

```
SELECT parseDateTimeBestEffort('2018-12-12 10:12:12')
AS parseDateTimeBestEffort;
```

Result:

```
parseDateTimeBestEffort
2018-12-12 10:12:12 |
```

Query:

```
SELECT parseDateTimeBestEffort('10 20:19');
```

Result:

```
parseDateTimeBestEffort('10 20:19')
2000-01-10 20:19:00 |
```

## See Also

- [ISO 8601 announcement by @xkcd](#)
- [RFC 1123](#)
- [toDate](#)
- [toDateTime](#)

## parseDateTimeBestEffortUS

This function is similar to [parseDateTimeBestEffort](#), the only difference is that this function prefers US date format (MM/DD/YYYY etc.) in case of ambiguity.

## Syntax

```
parseDateTimeBestEffortUS(time_string [, time_zone])
```

## Arguments

- `time_string` — String containing a date and time to convert. [String](#).
- `time_zone` — Time zone. The function parses `time_string` according to the time zone. [String](#).

## Supported non-standard formats

- A string containing 9..10 digit [unix timestamp](#).
- A string with a date and a time component: `YYYYMMDDhhmmss`, `MM/DD/YYYY hh:mm:ss`, `MM-DD-YY hh:mm`, `YYYY-MM-DD hh:mm:ss`, etc.
- A string with a date, but no time component: `YYYY`, `YYYYMM`, `YYYY*MM`, `MM/DD/YYYY`, `MM-DD-YY` etc.
- A string with a day and time: `DD`, `DD hh`, `DD hh:mm`. In this case, `YYYY-MM` are substituted as `2000-01`.
- A string that includes the date and time along with time zone offset information: `YYYY-MM-DD hh:mm:ss ±h:mm`, etc. For example, `2020-12-12 17:36:00 -5:00`.

## Returned value

- `time_string` converted to the [DateTime](#) data type.

## Examples

Query:

```
SELECT parseDateTimeBestEffortUS('09/12/2020 12:12:57')
AS parseDateTimeBestEffortUS;
```

Result:

```
parseDateTimeBestEffortUS→
2020-09-12 12:12:57 |
```

Query:

```
SELECT parseDateTimeBestEffortUS('09-12-2020 12:12:57')
AS parseDateTimeBestEffortUS;
```

Result:

```
parseDateTimeBestEffortUS→
2020-09-12 12:12:57 |
```

Query:

```
SELECT parseDateTimeBestEffortUS('09.12.2020 12:12:57')
AS parseDateTimeBestEffortUS;
```

Result:

```
parseDateTimeBestEffortUS
2020-09-12 12:12:57 |
```

## parseDateTimeBestEffortOrNull

## parseDateTime32BestEffortOrNull

Same as for [parseDateTimeBestEffort](#) except that it returns `NULL` when it encounters a date format that cannot be processed.

## parseDateTimeBestEffortOrZero

## parseDateTime32BestEffortOrZero

Same as for [parseDateTimeBestEffort](#) except that it returns zero date or zero date time when it encounters a date format that cannot be processed.

## parseDateTimeBestEffortUSOrNull

Same as [parseDateTimeBestEffortUS](#) function except that it returns `NULL` when it encounters a date format that cannot be processed.

### Syntax

```
parseDateTimeBestEffortUSOrNull(time_string[, time_zone])
```

### Parameters

- `time_string` — String containing a date or date with time to convert. The date must be in the US date format (MM/DD/YYYY, etc). [String](#).
- `time_zone` — [Timezone](#). The function parses `time_string` according to the timezone. Optional. [String](#).

### Supported non-standard formats

- A string containing 9..10 digit [unix timestamp](#).
- A string with a date and a time components: `YYYYMMDDhhmmss`, `MM/DD/YYYY hh:mm:ss`, `MM-DD-YY hh:mm`, `YYYY-MM-DD hh:mm:ss`, etc.
- A string with a date, but no time component: `YYYY`, `YYYYMM`, `YYYY*MM`, `MM/DD/YYYY`, `MM-DD-YY`, etc.
- A string with a day and time: `DD`, `DD hh`, `DD hh:mm`. In this case, `YYYY-MM` are substituted with `2000-01`.
- A string that includes date and time along with timezone offset information: `YYYY-MM-DD hh:mm:ss ±h:mm`, etc. For example, `2020-12-12 17:36:00 -5:00`.

### Returned values

- `time_string` converted to the [DateTime](#) data type.
- `NULL` if the input string cannot be converted to the [DateTime](#) data type.

### Examples

Query:

```
SELECT parseDateTimeBestEffortUSOrNull('02/10/2021 21:12:57') AS parseDateTimeBestEffortUSOrNull;
```

Result:

```
parseDateTimeBestEffortUSOrNull—  
2021-02-10 21:12:57 |
```

Query:

```
SELECT parseDateTimeBestEffortUSOrNull('02-10-2021 21:12:57 GMT', 'Europe/Moscow') AS parseDateTimeBestEffortUSOrNull;
```

Result:

```
parseDateTimeBestEffortUSOrNull—  
2021-02-11 00:12:57 |
```

Query:

```
SELECT parseDateTimeBestEffortUSOrNull('02.10.2021') AS parseDateTimeBestEffortUSOrNull;
```

Result:

```
parseDateTimeBestEffortUSOrNull—  
2021-02-10 00:00:00 |
```

Query:

```
SELECT parseDateTimeBestEffortUSOrNull('10.2021') AS parseDateTimeBestEffortUSOrNull;
```

Result:

```
parseDateTimeBestEffortUSOrNull—  
NULL |
```

## parseDateTimeBestEffortUSOrZero

Same as [parseDateTimeBestEffortUS](#) function except that it returns zero date (1970-01-01) or zero date with time (1970-01-01 00:00:00) when it encounters a date format that cannot be processed.

### Syntax

```
parseDateTimeBestEffortUSOrZero(time_string[, time_zone])
```

### Parameters

- `time_string` — String containing a date or date with time to convert. The date must be in the US date format (MM/DD/YYYY, etc). [String](#).

- `time_zone` — **Timezone**. The function parses `time_string` according to the timezone. Optional. **String**.

## Supported non-standard formats

- A string containing 9..10 digit **unix timestamp**.
- A string with a date and a time components: `YYYYMMDDhhmmss`, `MM/DD/YYYY hh:mm:ss`, `MM-DD-YY hh:mm`, `YYYY-MM-DD hh:mm:ss`, etc.
- A string with a date, but no time component: `YYYY`, `YYYYMM`, `YYYY*MM`, `MM/DD/YYYY`, `MM-DD-YY`, etc.
- A string with a day and time: `DD`, `DD hh`, `DD hh:mm`. In this case, `YYYY-MM` are substituted with `2000-01`.
- A string that includes date and time along with timezone offset information: `YYYY-MM-DD hh:mm:ss ±h:mm`, etc. For example, `2020-12-12 17:36:00 -5:00`.

## Returned values

- `time_string` converted to the **DateTime** data type.
- Zero date or zero date with time if the input string cannot be converted to the **DateTime** data type.

## Examples

Query:

```
SELECT parseDateTimeBestEffortUSOrZero('02/10/2021 21:12:57') AS parseDateTimeBestEffortUSOrZero;
```

Result:

```
parseDateTimeBestEffortUSOrZero→  
2021-02-10 21:12:57 |
```

Query:

```
SELECT parseDateTimeBestEffortUSOrZero('02-10-2021 21:12:57 GMT', 'Europe/Moscow') AS  
parseDateTimeBestEffortUSOrZero;
```

Result:

```
parseDateTimeBestEffortUSOrZero→  
2021-02-11 00:12:57 |
```

Query:

```
SELECT parseDateTimeBestEffortUSOrZero('02.10.2021') AS parseDateTimeBestEffortUSOrZero;
```

Result:

```
parseDateTimeBestEffortUSOrZero→  
2021-02-10 00:00:00 |
```

Query:

```
SELECT parseDateTimeBestEffortUSOrZero('02.2021') AS parseDateTimeBestEffortUSOrZero;
```

Result:

```
parseDateTimeBestEffortUSOrZero
1970-01-01 00:00:00 |
```

## parseDateTime64BestEffort

Same as [parseDateTimeBestEffort](#) function but also parse milliseconds and microseconds and returns [DateTime](#) data type.

### Syntax

```
parseDateTime64BestEffort(time_string [, precision [, time_zone]])
```

### Parameters

- `time_string` — String containing a date or date with time to convert. [String](#).
- `precision` — Required precision. `3` — for milliseconds, `6` — for microseconds. Default — `3`. Optional. [UInt8](#).
- `time_zone` — [Timezone](#). The function parses `time_string` according to the timezone. Optional. [String](#).

### Returned value

- `time_string` converted to the [DateTime](#) data type.

### Examples

Query:

```
SELECT parseDateTime64BestEffort('2021-01-01') AS a, toTypeName(a) AS t
UNION ALL
SELECT parseDateTime64BestEffort('2021-01-01 01:01:00.12346') AS a, toTypeName(a) AS t
UNION ALL
SELECT parseDateTime64BestEffort('2021-01-01 01:01:00.12346',6) AS a, toTypeName(a) AS t
UNION ALL
SELECT parseDateTime64BestEffort('2021-01-01 01:01:00.12346',3,'Europe/Moscow') AS a, toTypeName(a) AS t
FORMAT PrettyCompactMonoBlock;
```

Result:

```
a   t
2021-01-01 01:01:00.123000 | DateTime64(3)
2021-01-01 00:00:00.000000 | DateTime64(3)
2021-01-01 01:01:00.123460 | DateTime64(6)
2020-12-31 22:01:00.123000 | DateTime64(3, 'Europe/Moscow') |
```

## parseDateTime64BestEffortOrNull

Same as for [parseDateTime64BestEffort](#) except that it returns `NULL` when it encounters a date format that cannot be processed.

## parseDateTime64BestEffortOrZero

Same as for `parseDateTime64BestEffort` except that it returns zero date or zero date time when it encounters a date format that cannot be processed.

## toLowCardinality

Converts input parameter to the `LowCardianlity` version of same data type.

To convert data from the `LowCardinality` data type use the `CAST` function. For example, `CAST(x as String)`.

### Syntax

```
toLowCardinality(expr)
```

### Arguments

- `expr` — Expression resulting in one of the [supported data types](#).

### Returned values

- Result of `expr`.

Type: `LowCardinality(expr_result_type)`

### Example

Query:

```
SELECT toLowCardinality('1');
```

Result:

```
+-----+  
| 1 |  
+-----+
```

## toUnixTimestamp64Milli

## toUnixTimestamp64Micro

## toUnixTimestamp64Nano

Converts a `DateTime64` to a `Int64` value with fixed sub-second precision. Input value is scaled up or down appropriately depending on its precision.

### Note

The output value is a timestamp in UTC, not in the timezone of `DateTime64`.

### Syntax

```
toUnixTimestamp64Milli(value)
```

### Arguments

- `value` — `DateTime64` value with any precision.

## Returned value

- `value` converted to the `Int64` data type.

## Examples

Query:

```
WITH toDateTime64('2019-09-16 19:20:12.345678910', 6) AS dt64
SELECT toUnixTimestamp64Milli(dt64);
```

Result:

```
toUnixTimestamp64Milli(dt64)─
1568650812345 |
```

Query:

```
WITH toDateTime64('2019-09-16 19:20:12.345678910', 6) AS dt64
SELECT toUnixTimestamp64Nano(dt64);
```

Result:

```
toUnixTimestamp64Nano(dt64)─
1568650812345678000 |
```

## fromUnixTimestamp64Milli

## fromUnixTimestamp64Micro

## fromUnixTimestamp64Nano

Converts an `Int64` to a `DateTime64` value with fixed sub-second precision and optional timezone. Input value is scaled up or down appropriately depending on it's precision. Please note that input value is treated as UTC timestamp, not timestamp at given (or implicit) timezone.

## Syntax

```
fromUnixTimestamp64Milli(value [, ti])
```

## Arguments

- `value` — `Int64` value with any precision.
- `timezone` — `String` (optional) timezone name of the result.

## Returned value

- `value` converted to the `DateTime64` data type.

## Example

Query:

```
WITH CAST(1234567891011, 'Int64') AS i64
SELECT fromUnixTimestamp64Milli(i64, 'UTC');
```

Result:

```
fromUnixTimestamp64Milli(i64, 'UTC')—
2009-02-13 23:31:31.011 |
```

## formatRow

Converts arbitrary expressions into a string via given format.

### Syntax

```
formatRow(format, x, y, ...)
```

### Arguments

- `format` — Text format. For example, [CSV](#), [TSV](#).
- `x, y, ...` — Expressions.

### Returned value

- A formatted string (for text formats it's usually terminated with the new line character).

### Example

Query:

```
SELECT formatRow('CSV', number, 'good')
FROM numbers(3);
```

Result:

```
formatRow('CSV', number, 'good')—
0,"good"
| 1,"good"
| 2,"good"
```

## formatRowNoNewline

Converts arbitrary expressions into a string via given format. The function trims the last `\n` if any.

### Syntax

```
formatRowNoNewline(format, x, y, ...)
```

### Arguments

- `format` — Text format. For example, [CSV](#), [TSV](#).
- `x, y, ...` — Expressions.

## Returned value

- A formatted string.

## Example

Query:

```
SELECT formatRowNoNewline('CSV', number, 'good')
FROM numbers(3);
```

Result:

```
formatRowNoNewline('CSV', number, 'good')─
0,"good"
1,"good"
2,"good"
```

## snowflakeToDate

Extracts time from [Snowflake ID](#) as [DateTime](#) format.

### Syntax

```
snowflakeToDate(value [, time_zone])
```

### Parameters

- `value` — Snowflake ID. [Int64](#).
- `time_zone` — [Timezone](#). The function parses `time_string` according to the timezone. Optional. [String](#).

### Returned value

- Input value converted to the [DateTime](#) data type.

## Example

Query:

```
SELECT snowflakeToDate(CAST('1426860702823350272', 'Int64'), 'UTC');
```

Result:

```
snowflakeToDate(CAST('1426860702823350272', 'Int64'), 'UTC')─
2021-08-15 10:57:56 |
```

## snowflakeToDate64

Extracts time from [Snowflake ID](#) as [DateTime64](#) format.

### Syntax

```
snowflakeToDate64(value [, time_zone])
```

## Parameters

- `value` — Snowflake ID. [Int64](#).
- `time_zone` — [Timezone](#). The function parses `time_string` according to the timezone. Optional. [String](#).

## Returned value

- Input value converted to the [DateTime64](#) data type.

## Example

Query:

```
SELECT snowflakeToDateTime64(CAST('1426860802823350272', 'Int64'), 'UTC');
```

Result:

```
snowflakeToDateTime64(CAST('1426860802823350272', 'Int64'), 'UTC')—  
2021-08-15 10:58:19.841 |
```

## dateTimeToSnowflake

Converts [DateTime](#) value to the first [Snowflake ID](#) at the giving time.

## Syntax

```
dateTimeToSnowflake(value)
```

## Parameters

- `value` — Date and time. [DateTime](#).

## Returned value

- Input value converted to the [Int64](#) data type as the first Snowflake ID at that time.

## Example

Query:

```
WITH toDateTime('2021-08-15 18:57:56', 'Asia/Shanghai') AS dt SELECT dateTimeToSnowflake(dt);
```

Result:

```
dateTimeToSnowflake(dt)—  
1426860702823350272 |
```

## dateTime64ToSnowflake

Convert [DateTime64](#) to the first [Snowflake ID](#) at the giving time.

## Syntax

```
dateTime64ToSnowflake(value)
```

## Parameters

- `value` — Date and time. [DateTime64](#).

## Returned value

- Input value converted to the [Int64](#) data type as the first Snowflake ID at that time.

## Example

Query:

```
WITH toDateTime64('2021-08-15 18:57:56.492', 3, 'Asia/Shanghai') AS dt64 SELECT dateTime64ToSnowflake(dt64);
```

Result:

```
dateTime64ToSnowflake(dt64)  
1426860704886947840 |
```

# Functions for Working with Dates and Times

Support for time zones.

All functions for working with the date and time that have a logical use for the time zone can accept a second optional time zone argument. Example: Asia/Yekaterinburg. In this case, they use the specified time zone instead of the local (default) one.

```
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 |
```

## timeZone

Returns the timezone of the server.

If it is executed in the context of a distributed table, then it generates a normal column with values relevant to each shard. Otherwise it produces a constant value.

## Syntax

```
timeZone()
```

Alias: `timezone`.

## Returned value

- Timezone.

Type: [String](#).

# toTimeZone

Converts time or date and time to the specified time zone. The time zone is an attribute of the `Date` and `DateTime` data types. The internal value (number of seconds) of the table field or of the resultset's column does not change, the column's type changes and its string representation changes accordingly.

## Syntax

```
toTimezone(value, timezone)
```

Alias: `toTimezone`.

## Arguments

- `value` — Time or date and time. `DateTime64`.
- `timezone` — Timezone for the returned value. `String`.

## Returned value

- Date and time.

Type: `DateTime`.

## Example

Query:

```
SELECT toDate('2019-01-01 00:00:00', 'UTC') AS time_utc,
       toTypeName(time_utc) AS type_utc,
      .toInt32(time_utc) AS int32utc,
       toTimeZone(time_utc, 'Asia/Yekaterinburg') AS time_yekat,
       toTypeName(time_yekat) AS type_yekat,
      .toInt32(time_yekat) AS int32yekat,
       toTimeZone(time_utc, 'US/Samoa') AS time_samoa,
       toTypeName(time_samoa) AS type_samoa,
      .toInt32(time_samoa) AS int32samoa
FORMAT Vertical;
```

Result:

Row 1:

```
time_utc: 2019-01-01 00:00:00
type_utc: DateTime('UTC')
int32utc: 1546300800
time_yekat: 2019-01-01 05:00:00
type_yekat: DateTime('Asia/Yekaterinburg')
int32yekat: 1546300800
time_samoa: 2018-12-31 13:00:00
type_samoa: DateTime('US/Samoa')
int32samoa: 1546300800
```

`toTimeZone(time_utc, 'Asia/Yekaterinburg')` changes the `DateTime('UTC')` type to `DateTime('Asia/Yekaterinburg')`. The value (Unixtimestamp) 1546300800 stays the same, but the string representation (the result of the `toString()` function) changes from `time_utc: 2019-01-01 00:00:00` to `time_yekat: 2019-01-01 05:00:00`.

# timeZoneOf

Returns the timezone name of `DateTime` or `DateTime64` data types.

## Syntax

```
timeZoneOf(value)
```

Alias: `timezoneOf`.

## Arguments

- `value` — Date and time. [DateTime](#) or [DateTime64](#).

## Returned value

- Timezone name.

Type: [String](#).

## Example

Query:

```
SELECT timezoneOf(now());
```

Result:

```
timezoneOf(now())—  
Etc/UTC |
```

# timeZoneOffset

Returns a timezone offset in seconds from [UTC](#). The function takes into account [daylight saving time](#) and historical timezone changes at the specified date and time.

[IANA timezone database](#) is used to calculate the offset.

## Syntax

```
timeZoneOffset(value)
```

Alias: `timezoneOffset`.

## Arguments

- `value` — Date and time. [DateTime](#) or [DateTime64](#).

## Returned value

- Offset from UTC in seconds.

Type: [Int32](#).

## Example

Query:

```
SELECT toDateTime('2021-04-21 10:20:30', 'America/New_York') AS Time, toTypeName(Time) AS Type,  
timeZoneOffset(Time) AS Offset_in_seconds, (Offset_in_seconds / 3600) AS Offset_in_hours;
```

Result:

Time	Type	Offset_in_seconds	Offset_in_hours
2021-04-21 10:20:30	DateTime('America/New_York')	-14400	-4

## toYear

Converts a date or date with time to a UInt16 number containing the year number (AD).

Alias: `YEAR`.

## toQuarter

Converts a date or date with time to a UInt8 number containing the quarter number.

Alias: `QUARTER`.

## toMonth

Converts a date or date with time to a UInt8 number containing the month number (1-12).

Alias: `MONTH`.

## toDayOfYear

Converts a date or date with time to a UInt16 number containing the number of the day of the year (1-366).

Alias: `DAYOFYEAR`.

## toDayOfMonth

Converts a date or date with time to a UInt8 number containing the number of the day of the month (1-31).

Aliases: `DAYOFMONTH`, `DAY`.

## toDayOfWeek

Converts a date or date with time to a UInt8 number containing the number of the day of the week (Monday is 1, and Sunday is 7).

Alias: `DAYOFWEEK`.

## toHour

Converts a date with time to a UInt8 number containing the number of the hour in 24-hour time (0-23). This function assumes that if clocks are moved ahead, it is by one hour and occurs at 2 a.m., and if clocks are moved back, it is by one hour and occurs at 3 a.m. (which is not always true – even in Moscow the clocks were twice changed at a different time).

Alias: `HOUR`.

## toMinute

Converts a date with time to a UInt8 number containing the number of the minute of the hour (0-59).

Alias: `MINUTE`.

## toSecond

Converts a date with time to a UInt8 number containing the number of the second in the minute (0-59). Leap seconds are not accounted for.

Alias: `SECOND`.

## toUnixTimestamp

For `DateTime` argument: converts value to the number with type `UInt32` -- Unix Timestamp ([https://en.wikipedia.org/wiki/Unix\\_time](https://en.wikipedia.org/wiki/Unix_time)).

For `String` argument: converts the input string to the datetime according to the timezone (optional second argument, server timezone is used by default) and returns the corresponding unix timestamp.

### Syntax

```
toUnixTimestamp(datetime)
toUnixTimestamp(str, [timezone])
```

### Returned value

- Returns the unix timestamp.

Type: `UInt32`.

### Example

Query:

```
SELECT toUnixTimestamp('2017-11-05 08:07:47', 'Asia/Tokyo') AS unix_timestamp
```

Result:

```
unix_timestamp
1509836867 |
```

## Attention

The return type `toStartOf*` functions described below is `Date` or `DateTime`. Though these functions can take `DateTime64` as an argument, passing them a `DateTime64` that is out of the normal range (years 1925 - 2283) will give an incorrect result.

## toStartOfYear

Rounds down a date or date with time to the first day of the year.

Returns the date.

## toStartOfISOYear

Rounds down a date or date with time to the first day of ISO year.

Returns the date.

## toStartOfQuarter

Rounds down a date or date with time to the first day of the quarter.  
The first day of the quarter is either 1 January, 1 April, 1 July, or 1 October.  
Returns the date.

## toStartOfMonth

Rounds down a date or date with time to the first day of the month.  
Returns the date.

### Attention

The behavior of parsing incorrect dates is implementation specific. ClickHouse may return zero date, throw an exception or do “natural” overflow.

## toMonday

Rounds down a date or date with time to the nearest Monday.  
Returns the date.

## toStartOfWeek(t[,mode])

Rounds down a date or date with time to the nearest Sunday or Monday by mode.  
Returns the date.  
The mode argument works exactly like the mode argument to toWeek(). For the single-argument syntax, a mode value of 0 is used.

## toStartOfDay

Rounds down a date with time to the start of the day.

## toStartOfHour

Rounds down a date with time to the start of the hour.

## toStartOfMinute

Rounds down a date with time to the start of the minute.

## toStartOfSecond

Truncates sub-seconds.

### Syntax

```
toStartOfSecond(value, [timezone])
```

### Arguments

- `value` — Date and time. [DateTime64](#).
- `timezone` — [Timezone](#) for the returned value (optional). If not specified, the function uses the timezone of the `value` parameter. [String](#).

### Returned value

- Input value without sub-seconds.

Type: [DateTime64](#).

## Examples

Query without timezone:

```
WITH toDateTime64('2020-01-01 10:20:30.999', 3) AS dt64
SELECT toStartOfSecond(dt64);
```

Result:

```
toStartOfSecond(dt64)
2020-01-01 10:20:30.000 |
```

Query with timezone:

```
WITH toDateTime64('2020-01-01 10:20:30.999', 3) AS dt64
SELECT toStartOfSecond(dt64, 'Europe/Moscow');
```

Result:

```
toStartOfSecond(dt64, 'Europe/Moscow')
2020-01-01 13:20:30.000 |
```

## See also

- [Timezone](#) server configuration parameter.

## toStartOfFiveMinute

Rounds down a date with time to the start of the five-minute interval.

## toStartOfTenMinutes

Rounds down a date with time to the start of the ten-minute interval.

## toStartOfFifteenMinutes

Rounds down the date with time to the start of the fifteen-minute interval.

## toStartOfInterval(time\_or\_data, INTERVAL x unit [, time\_zone])

This is a generalization of other functions named `toStartOf*`. For example,  
`toStartOfInterval(t, INTERVAL 1 year)` returns the same as `toStartOfYear(t)`,  
`toStartOfInterval(t, INTERVAL 1 month)` returns the same as `toStartOfMonth(t)`,  
`toStartOfInterval(t, INTERVAL 1 day)` returns the same as `toStartOfDay(t)`,  
`toStartOfInterval(t, INTERVAL 15 minute)` returns the same as `toStartOfFifteenMinutes(t)` etc.

## toTime

Converts a date with time to a certain fixed date, while preserving the time.

## toRelativeYearNum

Converts a date with time or date to the number of the year, starting from a certain fixed point in the past.

## toRelativeQuarterNum

Converts a date with time or date to the number of the quarter, starting from a certain fixed point in the past.

## toRelativeMonthNum

Converts a date with time or date to the number of the month, starting from a certain fixed point in the past.

## toRelativeWeekNum

Converts a date with time or date to the number of the week, starting from a certain fixed point in the past.

## toRelativeDayNum

Converts a date with time or date to the number of the day, starting from a certain fixed point in the past.

## toRelativeHourNum

Converts a date with time or date to the number of the hour, starting from a certain fixed point in the past.

## toRelativeMinuteNum

Converts a date with time or date to the number of the minute, starting from a certain fixed point in the past.

## toRelativeSecondNum

Converts a date with time or date to the number of the second, starting from a certain fixed point in the past.

## toISOYear

Converts a date or date with time to a UInt16 number containing the ISO Year number.

## toISOWeek

Converts a date or date with time to a UInt8 number containing the ISO Week number.

## toWeek(date[,mode])

This function returns the week number for date or datetime. The two-argument form of toWeek() enables you to specify whether the week starts on Sunday or Monday and whether the return value should be in the range from 0 to 53 or from 1 to 53. If the mode argument is omitted, the default mode is 0.

toISOWeek() is a compatibility function that is equivalent to toWeek(date,3).

The following table describes how the mode argument works.

Mode	First day of week	Range	Week 1 is the first week ...
0	Sunday	0-53	with a Sunday in this year
1	Monday	0-53	with 4 or more days this year

Mode	First day of week	Range	Week 1 is the first week ...
2	Sunday	1-53	with a Sunday in this year
3	Monday	1-53	with 4 or more days this year
4	Sunday	0-53	with 4 or more days this year
5	Monday	0-53	with a Monday in this year
6	Sunday	1-53	with 4 or more days this year
7	Monday	1-53	with a Monday in this year
8	Sunday	1-53	contains January 1
9	Monday	1-53	contains January 1

For mode values with a meaning of “with 4 or more days this year,” weeks are numbered according to ISO 8601:1988:

- If the week containing January 1 has 4 or more days in the new year, it is week 1.
- Otherwise, it is the last week of the previous year, and the next week is week 1.

For mode values with a meaning of “contains January 1”, the week contains January 1 is week 1. It does not matter how many days in the new year the week contained, even if it contained only one day.

```
toWeek(date, [, mode][, Timezone])
```

## Arguments

- `date` – Date or DateTime.
- `mode` – Optional parameter, Range of values is [0,9], default is 0.
- `Timezone` – Optional parameter, it behaves like any other conversion function.

## Example

```
SELECT toDate('2016-12-27') AS date, toWeek(date) AS week0, toWeek(date,1) AS week1, toWeek(date,9) AS week9;
```

date	week0	week1	week9
2016-12-27	52	52	1

## toYearWeek(date[,mode])

Returns year and week for a date. The year in the result may be different from the year in the date argument for the first and the last week of the year.

The mode argument works exactly like the mode argument to `toWeek()`. For the single-argument syntax, a mode value of 0 is used.

`toISOYear()` is a compatibility function that is equivalent to `intDiv(toYearWeek(date,3),100)`.

## Example

```
SELECT toDate('2016-12-27') AS date, toYearWeek(date) AS yearWeek0, toYearWeek(date,1) AS yearWeek1,  
toYearWeek(date,9) AS yearWeek9;
```

date	yearWeek0	yearWeek1	yearWeek9
2016-12-27	201652	201652	201701

## date\_trunc

Truncates date and time data to the specified part of date.

### Syntax

```
date_trunc(unit, value[, timezone])
```

Alias: `dateTrunc`.

### Arguments

- `unit` — The type of interval to truncate the result. [String Literal](#).

Possible values:

- `second`
- `minute`
- `hour`
- `day`
- `week`
- `month`
- `quarter`
- `year`

- `value` — Date and time. [DateTime](#) or [DateTime64](#).

- `timezone` — [Timezone name](#) for the returned value (optional). If not specified, the function uses the timezone of the `value` parameter. [String](#).

### Returned value

- Value, truncated to the specified part of date.

Type: [Datetime](#).

## Example

Query without timezone:

```
SELECT now(), date_trunc('hour', now());
```

Result:

```
now()—date_trunc('hour', now())—  
2020-09-28 10:40:45 | 2020-09-28 10:00:00 |
```

Query with the specified timezone:

```
SELECT now(), date_trunc('hour', now(), 'Europe/Moscow');
```

Result:

```
now()—date_trunc('hour', now(), 'Europe/Moscow')—  
2020-09-28 10:46:26 | 2020-09-28 13:00:00 |
```

## See Also

- [toStartOfInterval](#)

## date\_add

Adds the time interval or date interval to the provided date or date with time.

### Syntax

```
date_add(unit, value, date)
```

Aliases: `dateAdd`, `DATE_ADD`.

### Arguments

- `unit` — The type of interval to add. [String](#).

Possible values:

- `second`
  - `minute`
  - `hour`
  - `day`
  - `week`
  - `month`
  - `quarter`
  - `year`
- `value` — Value of interval to add. [Int](#).
  - `date` — The date or date with time to which `value` is added. [Date](#) or [DateTime](#).

### Returned value

Date or date with time obtained by adding `value`, expressed in `unit`, to `date`.

Type: [Date](#) or [DateTime](#).

## Example

Query:

```
SELECT date_add(YEAR, 3, toDate('2018-01-01'));
```

Result:

```
plus(toDate('2018-01-01'), toIntervalYear(3))-->  
2021-01-01 |
```

## date\_diff

Returns the difference between two dates or dates with time values.

### Syntax

```
date_diff('unit', startdate, enddate, [timezone])
```

Aliases: `dateDiff`, `DATE_DIFF`.

### Arguments

- `unit` — The type of interval for result. [String](#).

Possible values:

- `second`
- `minute`
- `hour`
- `day`
- `week`
- `month`
- `quarter`
- `year`

- `startdate` — The first time value to subtract (the subtrahend). [Date](#) or [DateTime](#).

- `enddate` — The second time value to subtract from (the minuend). [Date](#) or [DateTime](#).

- `timezone` — [Timezone name](#) (optional). If specified, it is applied to both `startdate` and `enddate`. If not specified, timezones of `startdate` and `enddate` are used. If they are not the same, the result is unspecified. [String](#).

### Returned value

Difference between `enddate` and `startdate` expressed in `unit`.

Type: [Int](#).

## Example

Query:

```
SELECT dateDiff('hour', toDateTime('2018-01-01 22:00:00'), toDateTime('2018-01-02 23:00:00'));
```

Result:

```
dateDiff('hour', toDateTime('2018-01-01 22:00:00'), toDateTime('2018-01-02 23:00:00'))—  
25 |
```

## date\_sub

Subtracts the time interval or date interval from the provided date or date with time.

### Syntax

```
date_sub(unit, value, date)
```

Aliases: `dateSub`, `DATE_SUB`.

### Arguments

- `unit` — The type of interval to subtract. [String](#).

Possible values:

- `second`
- `minute`
- `hour`
- `day`
- `week`
- `month`
- `quarter`
- `year`

- `value` — Value of interval to subtract. [Int](#).

- `date` — The date or date with time from which `value` is subtracted. [Date](#) or [DateTime](#).

### Returned value

Date or date with time obtained by subtracting `value`, expressed in `unit`, from `date`.

Type: [Date](#) or [DateTime](#).

### Example

Query:

```
SELECT date_sub(YEAR, 3, toDate('2018-01-01'));
```

Result:

```
minus(toDate('2018-01-01'), toIntervalYear(3))—  
2015-01-01 |
```

## timestamp\_add

Adds the specified time value with the provided date or date time value.

### Syntax

```
timestamp_add(date, INTERVAL value unit)
```

Aliases: `timeStampAdd`, `TIMESTAMP_ADD`.

### Arguments

- `date` — Date or date with time. [Date](#) or [DateTime](#).
- `value` — Value of interval to add. [Int](#).
- `unit` — The type of interval to add. [String](#).

Possible values:

- `second`
- `minute`
- `hour`
- `day`
- `week`
- `month`
- `quarter`
- `year`

### Returned value

Date or date with time with the specified value expressed in `unit` added to date.

Type: [Date](#) or [DateTime](#).

### Example

Query:

```
select timestamp_add(toDate('2018-01-01'), INTERVAL 3 MONTH);
```

Result:

```
plus(toDate('2018-01-01'), toIntervalMonth(3))—  
2018-04-01 |
```

## timestamp\_sub

Subtracts the time interval from the provided date or date with time.

## Syntax

```
timestamp_sub(unit, value, date)
```

Aliases: `timeStampSub`, `TIMESTAMP_SUB`.

## Arguments

- `unit` — The type of interval to subtract. [String](#).

Possible values:

- `second`
- `minute`
- `hour`
- `day`
- `week`
- `month`
- `quarter`
- `year`

- `value` — Value of interval to subtract. [Int](#).

- `date` — Date or date with time. [Date](#) or [DateTime](#).

## Returned value

Date or date with time obtained by subtracting `value`, expressed in `unit`, from `date`.

Type: [Date](#) or [DateTime](#).

## Example

Query:

```
select timestamp_sub(MONTH, 5, toDateTime('2018-12-18 01:02:03'));
```

Result:

```
minus(toDateTime('2018-12-18 01:02:03'), toIntervalMonth(5))—  
2018-07-18 01:02:03 |
```

## NOW

Returns the current date and time.

## Syntax

```
now([timezone])
```

## Arguments

- `timezone` — **Timezone name** for the returned value (optional). **String**.

## Returned value

- Current date and time.

Type: **Datetime**.

## Example

Query without timezone:

```
SELECT now();
```

Result:

```
now()  
2020-10-17 07:42:09 |
```

Query with the specified timezone:

```
SELECT now('Europe/Moscow');
```

Result:

```
now('Europe/Moscow')  
2020-10-17 10:42:23 |
```

## today

Accepts zero arguments and returns the current date at one of the moments of request execution.  
The same as ‘`toDate(now())`’.

## yesterday

Accepts zero arguments and returns yesterday’s date at one of the moments of request execution.  
The same as ‘`today() - 1`’.

## timeSlot

Rounds the time to the half hour.

This function is specific to Yandex.Metrica, since half an hour is the minimum amount of time for breaking a session into two sessions if a tracking tag shows a single user’s consecutive pageviews that differ in time by strictly more than this amount. This means that tuples (the tag ID, user ID, and time slot) can be used to search for pageviews that are included in the corresponding session.

## toYYYYMM

Converts a date or date with time to a UInt32 number containing the year and month number (YYYY \* 100 + MM).

## toYYYYMMDD

Converts a date or date with time to a UInt32 number containing the year and month number (YYYY \* 10000 + MM \* 100 + DD).

## toYYYYMMDDhhmmss

Converts a date or date with time to a UInt64 number containing the year and month number (YYYY \* 10000000000 + MM \* 100000000 + DD \* 1000000 + hh \* 10000 + mm \* 100 + ss).

## addYears, addMonths, addWeeks, addDays, addHours, addMinutes, addSeconds, addQuarters

Function adds a Date/DateTime interval to a Date/DateTime and then return the Date/DateTime. For example:

```
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, subtractHours, subtractMinutes, subtractSeconds, subtractQuarters

Function subtract a Date/DateTime interval to a Date/DateTime and then return the Date/DateTime. For example:

```
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

## timeSlots(StartTime, Duration[, Size])

For a time interval starting at 'StartTime' and continuing for 'Duration' seconds, it returns an array of moments in time, consisting of points from this interval rounded down to the 'Size' in seconds. 'Size' is an optional parameter: a constant UInt32, set to 1800 by default.

For example, `timeSlots(toDateTime('2012-01-01 12:20:00'), 600) = [toDateTime('2012-01-01 12:00:00'), toDateTime('2012-01-01 12:30:00')]`.

This is necessary for searching for pageviews in the corresponding session.

## formatDateTime

Formats a Time according to the given Format string. Format is a constant expression, so you cannot have multiple formats for a single result column.

## Syntax

```
formatDateTime(Time, Format[, Timezone])
```

### Returned value(s)

Returns time and date values according to the determined format.

### Replacement fields

Using replacement fields, you can define a pattern for the resulting string. "Example" column shows formatting result for 2018-01-02 22:33:44.

Placeholder	Description	Example
%C	year divided by 100 and truncated to integer (00-99)	20
%d	day of the month, zero-padded (01-31)	02
%D	Short MM/DD/YY date, equivalent to %m/%d/%y	01/02/18
%e	day of the month, space-padded ( 1-31)	2
%F	short YYYY-MM-DD date, equivalent to %Y-%m-%d	2018-01-02
%G	four-digit year format for ISO week number, calculated from the week-based year <b>defined by the ISO 8601 standard</b> , normally useful only with %V	2018
%g	two-digit year format, aligned to ISO 8601, abbreviated from four-digit notation	18
%H	hour in 24h format (00-23)	22
%I	hour in 12h format (01-12)	10
%j	day of the year (001-366)	002
%m	month as a decimal number (01-12)	01
%M	minute (00-59)	33
%n	new-line character ('')	
%p	AM or PM designation	PM
%Q	Quarter (1-4)	1
%R	24-hour HH:MM time, equivalent to %H:%M	22:33
%S	second (00-59)	44
%t	horizontal-tab character ('')	

Placeholder	Description	Example
%T	ISO 8601 time format (HH:MM:SS), equivalent to %H:%M:%S	22:33:44
%u	ISO 8601 weekday as number with Monday as 1 (1-7)	2
%V	ISO 8601 week number (01-53)	01
%w	weekday as a decimal number with Sunday as 0 (0-6)	2
%y	Year, last two digits (00-99)	18
%Y	Year	2018
%%	a % sign	%

## Example

Query:

```
SELECT formatDateTime(toDate('2010-01-04'), '%g')
```

Result:

```
formatDateTime(toDate('2010-01-04'), '%g')—  
10 |
```

## dateName

Returns specified part of date.

### Syntax

```
dateName(date_part, date)
```

### Arguments

- **date\_part** — Date part. Possible values: 'year', 'quarter', 'month', 'week', 'dayofyear', 'day', 'weekday', 'hour', 'minute', 'second'. **String**.
- **date** — Date. **Date**, **DateTime** or **DateTime64**.
- **timezone** — Timezone. Optional. **String**.

### Returned value

- The specified part of date.

Type: **String**

## Example

Query:

```
WITH toDateTime('2021-04-14 11:22:33') AS date_value
SELECT dateName('year', date_value), dateName('month', date_value), dateName('day', date_value);
```

Result:

dateName('year', date_value)	—	dateName('month', date_value)	—	dateName('day', date_value)	—
2021	April	14			

## FROM\_UNIXTIME

Function converts Unix timestamp to a calendar date and a time of a day. When there is only a single argument of [Integer](#) type, it acts in the same way as [toDateTime](#) and return [DateTime](#) type.

### Example:

Query:

```
SELECT FROM_UNIXTIME(423543535);
```

Result:

FROM_UNIXTIME(423543535)	—
1983-06-04 10:58:55	

When there are two arguments: first is an [Integer](#) or [DateTime](#), second is a constant format string — it acts in the same way as [formatDateTime](#) and return [String](#) type.

For example:

```
SELECT FROM_UNIXTIME(1234334543, '%Y-%m-%d %R:%S') AS DateTime;
```

DateTime	—
2009-02-11 14:42:23	

## toModifiedJulianDay

Converts a [Proleptic Gregorian calendar](#) date in text form YYYY-MM-DD to a [Modified Julian Day](#) number in Int32. This function supports date from 0000-01-01 to 9999-12-31. It raises an exception if the argument cannot be parsed as a date, or the date is invalid.

### Syntax

```
toModifiedJulianDay(date)
```

### Arguments

- `date` — Date in text form. [String](#) or [FixedString](#).

### Returned value

- Modified Julian Day number.

Type: [Int32](#).

## Example

Query:

```
SELECT toModifiedJulianDay('2020-01-01');
```

Result:

```
toModifiedJulianDay('2020-01-01')—  
      58849 |
```

## toModifiedJulianDayOrNull

Similar to [toModifiedJulianDay\(\)](#), but instead of raising exceptions it returns [NULL](#).

## Syntax

```
toModifiedJulianDayOrNull(date)
```

## Arguments

- **date** — Date in text form. [String](#) or [FixedString](#).

## Returned value

- Modified Julian Day number.

Type: [Nullable\(Int32\)](#).

## Example

Query:

```
SELECT toModifiedJulianDayOrNull('2020-01-01');
```

Result:

```
toModifiedJulianDayOrNull('2020-01-01')—  
      58849 |
```

## fromModifiedJulianDay

Converts a [Modified Julian Day](#) number to a [Proleptic Gregorian calendar](#) date in text form `YYYY-MM-DD`. This function supports day number from -678941 to 2973119 (which represent 0000-01-01 and 9999-12-31 respectively). It raises an exception if the day number is outside of the supported range.

## Syntax

```
fromModifiedJulianDay(day)
```

## Arguments

- `day` — Modified Julian Day number. [Any integral types](#).

## Returned value

- Date in text form.

Type: [String](#)

## Example

Query:

```
SELECT fromModifiedJulianDay(58849);
```

Result:

```
fromModifiedJulianDay(58849)─  
2020-01-01 ┌─────────┐
```

## fromModifiedJulianDayOrNull

Similar to [fromModifiedJulianDayOrNull\(\)](#), but instead of raising exceptions it returns [NULL](#).

## Syntax

```
fromModifiedJulianDayOrNull(day)
```

## Arguments

- `day` — Modified Julian Day number. [Any integral types](#).

## Returned value

- Date in text form.

Type: [Nullable\(String\)](#)

## Example

Query:

```
SELECT fromModifiedJulianDayOrNull(58849);
```

Result:

```
fromModifiedJulianDayOrNull(58849)─  
2020-01-01 ┌─────────┐
```

# Functions for Working with Strings

## Note

Functions for [searching](#) and [replacing](#) in strings are described separately.

# empty

Checks whether the input string is empty.

## Syntax

```
empty(x)
```

A string is considered non-empty if it contains at least one byte, even if this is a space or a null byte.

The function also works for [arrays](#) or [UUID](#).

## Arguments

- `x` — Input value. [String](#).

## Returned value

- Returns `1` for an empty string or `0` for a non-empty string.

Type: [UInt8](#).

## Example

Query:

```
SELECT empty('');
```

Result:

```
+-----+  
| empty('') |  
+-----+  
| 1 |
```

# notEmpty

Checks whether the input string is non-empty.

## Syntax

```
notEmpty(x)
```

A string is considered non-empty if it contains at least one byte, even if this is a space or a null byte.

The function also works for [arrays](#) or [UUID](#).

## Arguments

- `x` — Input value. [String](#).

## Returned value

- Returns `1` for a non-empty string or `0` for an empty string string.

Type: [UInt8](#).

## Example

Query:

```
SELECT notEmpty('text');
```

Result:

```
notEmpty('text')  
1 |
```

## length

Returns the length of a string in bytes (not in characters, and not in code points).

The result type is UInt64.

The function also works for arrays.

## lengthUTF8

Returns the length of a string in Unicode code points (not in characters), assuming that the string contains a set of bytes that make up UTF-8 encoded text. If this assumption is not met, it returns some result (it does not throw an exception).

The result type is UInt64.

## char\_length, CHAR\_LENGTH

Returns the length of a string in Unicode code points (not in characters), assuming that the string contains a set of bytes that make up UTF-8 encoded text. If this assumption is not met, it returns some result (it does not throw an exception).

The result type is UInt64.

## character\_length, CHARACTER\_LENGTH

Returns the length of a string in Unicode code points (not in characters), assuming that the string contains a set of bytes that make up UTF-8 encoded text. If this assumption is not met, it returns some result (it does not throw an exception).

The result type is UInt64.

## leftPad

Pads the current string from the left with spaces or a specified string (multiple times, if needed) until the resulting string reaches the given length. Similarly to the MySQL LPAD function.

### Syntax

```
leftPad('string', 'length'[, 'pad_string'])
```

### Arguments

- `string` — Input string that needs to be padded. [String](#).
- `length` — The length of the resulting string. [UInt](#). If the value is less than the input string length, then the input string is returned as-is.
- `pad_string` — The string to pad the input string with. [String](#). Optional. If not specified, then the input string is padded with spaces.

### Returned value

- The resulting string of the given length.

Type: **String**.

### Example

Query:

```
SELECT leftPad('abc', 7, '*'), leftPad('def', 7);
```

Result:

```
leftPad('abc', 7, '*') └─ leftPad('def', 7) └  
*****abc           |   def      |
```

## leftPadUTF8

Pads the current string from the left with spaces or a specified string (multiple times, if needed) until the resulting string reaches the given length. Similarly to the MySQL LPAD function. While in the [leftPad](#) function the length is measured in bytes, here in the `leftPadUTF8` function it is measured in code points.

### Syntax

```
leftPadUTF8('string', 'length'[, 'pad_string'])
```

### Arguments

- `string` — Input string that needs to be padded. [String](#).
- `length` — The length of the resulting string. [UInt](#). If the value is less than the input string length, then the input string is returned as-is.
- `pad_string` — The string to pad the input string with. [String](#). Optional. If not specified, then the input string is padded with spaces.

### Returned value

- The resulting string of the given length.

Type: **String**.

### Example

Query:

```
SELECT leftPadUTF8('абвг', 7, '*'), leftPadUTF8('дежз', 7);
```

Result:

```
leftPadUTF8('абвг', 7, '*') └─ leftPadUTF8('дежз', 7) └  
***абвг           |   дежз      |
```

## rightPad

Pads the current string from the right with spaces or a specified string (multiple times, if needed) until the resulting string reaches the given length. Similarly to the MySQL RPAD function.

## Syntax

```
rightPad('string', 'length'[, 'pad_string'])
```

## Arguments

- **string** — Input string that needs to be padded. [String](#).
- **length** — The length of the resulting string. [UInt](#). If the value is less than the input string length, then the input string is returned as-is.
- **pad\_string** — The string to pad the input string with. [String](#). Optional. If not specified, then the input string is padded with spaces.

## Returned value

- The resulting string of the given length.

Type: [String](#).

## Example

Query:

```
SELECT rightPad('abc', 7, '*'), rightPad('abc', 7);
```

Result:

```
rightPad('abc', 7, '*') rightPad('abc', 7)
abc****      | abc      |
```

## rightPadUTF8

Pads the current string from the right with spaces or a specified string (multiple times, if needed) until the resulting string reaches the given length. Similarly to the MySQL RPAD function. While in the [rightPad](#) function the length is measured in bytes, here in the [rightPadUTF8](#) function it is measured in code points.

## Syntax

```
rightPadUTF8('string','length'[, 'pad_string'])
```

## Arguments

- **string** — Input string that needs to be padded. [String](#).
- **length** — The length of the resulting string. [UInt](#). If the value is less than the input string length, then the input string is returned as-is.
- **pad\_string** — The string to pad the input string with. [String](#). Optional. If not specified, then the input string is padded with spaces.

## Returned value

- The resulting string of the given length.

Type: **String**.

## Example

Query:

```
SELECT rightPadUTF8('абвг', 7, '*'), rightPadUTF8('абвг', 7);
```

Result:

```
+-----+-----+
| rightPadUTF8('абвг', 7, '*') | rightPadUTF8('абвг', 7) |
|    абвг***                 |    абвг               |
+-----+-----+
```

## lower, lcase

Converts ASCII Latin symbols in a string to lowercase.

## upper, ucase

Converts ASCII Latin symbols in a string to uppercase.

## lowerUTF8

Converts a string to lowercase, assuming the string contains a set of bytes that make up a UTF-8 encoded text.

It does not detect the language. So for Turkish the result might not be exactly correct.

If the length of the UTF-8 byte sequence is different for upper and lower case of a code point, the result may be incorrect for this code point.

If the string contains a set of bytes that is not UTF-8, then the behavior is undefined.

## upperUTF8

Converts a string to uppercase, assuming the string contains a set of bytes that make up a UTF-8 encoded text.

It does not detect the language. So for Turkish the result might not be exactly correct.

If the length of the UTF-8 byte sequence is different for upper and lower case of a code point, the result may be incorrect for this code point.

If the string contains a set of bytes that is not UTF-8, then the behavior is undefined.

## isValidUTF8

Returns 1, if the set of bytes is valid UTF-8 encoded, otherwise 0.

## toValidUTF8

Replaces invalid UTF-8 characters by the ♦ (U+FFFD) character. All running in a row invalid characters are collapsed into the one replacement character.

```
toValidUTF8(input_string)
```

## Arguments

- input\_string — Any set of bytes represented as the **String** data type object.

Returned value: Valid UTF-8 string.

## Example

```
SELECT toValidUTF8('\x61\xF0\x80\x80\x80b');
```

```
toValidUTF8('a♦♦♦♦b')  
a♦b
```

## repeat

Repeats a string as many times as specified and concatenates the replicated values as a single string.

Alias: `REPEAT`.

### Syntax

```
repeat(s, n)
```

### Arguments

- `s` — The string to repeat. [String](#).
- `n` — The number of times to repeat the string. [UInt](#).

### Returned value

The single string, which contains the string `s` repeated `n` times. If `n < 1`, the function returns empty string.

Type: [String](#).

## Example

Query:

```
SELECT repeat('abc', 10);
```

Result:

```
repeat('abc', 10)  
abcabcabcabcabcabcabcabcabc |
```

## reverse

Reverses the string (as a sequence of bytes).

## reverseUTF8

Reverses a sequence of Unicode code points, assuming that the string contains a set of bytes representing a UTF-8 text. Otherwise, it does something else (it does not throw an exception).

## format(pattern, s0, s1, ...)

Formatting constant pattern with the string listed in the arguments. `pattern` is a simplified Python format pattern. Format string contains “replacement fields” surrounded by curly braces `{}`. Anything that is not contained in braces is considered literal text, which is copied unchanged to the output. If you need to include a brace character in the literal text, it can be escaped by doubling: `{}{` and `}}`. Field names can be numbers (starting from zero) or empty (then they are treated as consequence numbers).

```
SELECT format('{1} {0} {1}', 'World', 'Hello')
```

```
format('{1} {0} {1}', 'World', 'Hello')  
Hello World Hello |
```

```
SELECT format('{} {}', 'Hello', 'World')
```

```
format('{} {}'.format('Hello', 'World'))  
Hello World |
```

## concat

Concatenates the strings listed in the arguments, without a separator.

### Syntax

```
concat(s1, s2, ...)
```

### Arguments

Values of type String or FixedString.

### Returned values

Returns the String that results from concatenating the arguments.

If any of argument values is NULL, concat returns NULL.

### Example

Query:

```
SELECT concat('Hello, ', 'World!');
```

Result:

```
concat('Hello, ', 'World!')  
Hello, World! |
```

## concatAssumeInjective

Same as `concat`, the difference is that you need to ensure that `concat(s1, s2, ...) → sn` is injective, it will be used for optimization of GROUP BY.

The function is named “injective” if it always returns different result for different values of arguments. In other words: different arguments never yield identical result.

## Syntax

```
concatAssumeInjective(s1, s2, ...)
```

## Arguments

Values of type String or FixedString.

## Returned values

Returns the String that results from concatenating the arguments.

If any of argument values is `NULL`, `concatAssumeInjective` returns `NULL`.

## Example

Input table:

```
CREATE TABLE key_val(`key1` String, `key2` String, `value` UInt32) ENGINE = TinyLog;
INSERT INTO key_val VALUES ('Hello, ','World',1), ('Hello, ','World',2), ('Hello, ','World!',3), ('Hello',' ', World!',2);
SELECT * from key_val;
```

key1	key2	value
Hello,	World	1
Hello,	World	2
Hello,	World!	3
Hello	, World!	2

Query:

```
SELECT concat(key1, key2), sum(value) FROM key_val GROUP BY concatAssumeInjective(key1, key2);
```

Result:

concat(key1, key2)	sum(value)
Hello, World!	3
Hello, World!	2
Hello, World	3

## substring(s, offset, length), mid(s, offset, length), substr(s, offset, length)

Returns a substring starting with the byte from the ‘offset’ index that is ‘length’ bytes long. Character indexing starts from one (as in standard SQL). The ‘offset’ and ‘length’ arguments must be constants.

## substringUTF8(s, offset, length)

The same as ‘substring’, but for Unicode code points. Works under the assumption that the string contains a set of bytes representing a UTF-8 encoded text. If this assumption is not met, it returns some result (it does not throw an exception).

## appendTrailingCharIfAbsent(s, c)

If the 's' string is non-empty and does not contain the 'c' character at the end, it appends the 'c' character to the end.

## convertCharset(s, from, to)

Returns the string 's' that was converted from the encoding in 'from' to the encoding in 'to'.

## base64Encode(s)

Encodes 's' string into base64

Alias: TO\_BASE64.

## base64Decode(s)

Decode base64-encoded string 's' into original string. In case of failure raises an exception.

Alias: FROM\_BASE64.

## tryBase64Decode(s)

Similar to base64Decode, but in case of error an empty string would be returned.

## endsWith(s, suffix)

Returns whether to end with the specified suffix. Returns 1 if the string ends with the specified suffix, otherwise it returns 0.

## startsWith(str, prefix)

Returns 1 whether string starts with the specified prefix, otherwise it returns 0.

```
SELECT startsWith('Spider-Man', 'Spi');
```

### Returned values

- 1, if the string starts with the specified prefix.
- 0, if the string does not start with the specified prefix.

### Example

Query:

```
SELECT startsWith('Hello, world!', 'He');
```

Result:

```
startsWith('Hello, world!', 'He')—  
    1 |
```

## trim

Removes all specified characters from the start or end of a string.

By default removes all consecutive occurrences of common whitespace (ASCII character 32) from both ends of a string.

## Syntax

```
trim([[LEADING|TRAILING|BOTH] trim_character FROM] input_string)
```

## Arguments

- `trim_character` — Specified characters for trim. [String](#).
- `input_string` — String for trim. [String](#).

## Returned value

A string without leading and (or) trailing specified characters.

Type: [String](#).

## Example

Query:

```
SELECT trim(BOTH '()' FROM '(Hello, world!)');
```

Result:

```
trim(BOTH '()' FROM '(Hello, world!)')—  
Hello, world!
```

## trimLeft

Removes all consecutive occurrences of common whitespace (ASCII character 32) from the beginning of a string. It does not remove other kinds of whitespace characters (tab, no-break space, etc.).

## Syntax

```
trimLeft(input_string)
```

Alias: [ltrim\(input\\_string\)](#).

## Arguments

- `input_string` — string to trim. [String](#).

## Returned value

A string without leading common whitespaces.

Type: [String](#).

## Example

Query:

```
SELECT trimLeft('Hello, world!');
```

Result:

```
trimLeft('Hello, world! ')  
Hello, world!
```

## trimRight

Removes all consecutive occurrences of common whitespace (ASCII character 32) from the end of a string. It does not remove other kinds of whitespace characters (tab, no-break space, etc.).

### Syntax

```
trimRight(input_string)
```

Alias: `rtrim(input_string)`.

### Arguments

- `input_string` — string to trim. [String](#).

### Returned value

A string without trailing common whitespaces.

Type: `String`.

### Example

Query:

```
SELECT trimRight('Hello, world!');
```

Result:

```
trimRight('Hello, world! ')  
Hello, world!
```

## trimBoth

Removes all consecutive occurrences of common whitespace (ASCII character 32) from both ends of a string. It does not remove other kinds of whitespace characters (tab, no-break space, etc.).

### Syntax

```
trimBoth(input_string)
```

Alias: `trim(input_string)`.

### Arguments

- `input_string` — string to trim. [String](#).

### Returned value

A string without leading and trailing common whitespaces.

Type: `String`.

## Example

Query:

```
SELECT trimBoth('  Hello, world!  ');
```

Result:

```
trimBoth('  Hello, world!  ')  
Hello, world!
```

## CRC32(s)

Returns the CRC32 checksum of a string, using CRC-32-IEEE 802.3 polynomial and initial value 0xffffffff (zlib implementation).

The result type is UInt32.

## CRC32IEEE(s)

Returns the CRC32 checksum of a string, using CRC-32-IEEE 802.3 polynomial.

The result type is UInt32.

## CRC64(s)

Returns the CRC64 checksum of a string, using CRC-64-ECMA polynomial.

The result type is UInt64.

## normalizeQuery

Replaces literals, sequences of literals and complex aliases with placeholders.

### Syntax

```
normalizeQuery(x)
```

### Arguments

- `x` — Sequence of characters. [String](#).

### Returned value

- Sequence of characters with placeholders.

Type: [String](#).

## Example

Query:

```
SELECT normalizeQuery('[1, 2, 3, x]') AS query;
```

Result:

```
query  
[?.., x] |
```

## normalizedQueryHash

Returns identical 64bit hash values without the values of literals for similar queries. It helps to analyze query log.

### Syntax

```
normalizedQueryHash(x)
```

### Arguments

- x — Sequence of characters. [String](#).

### Returned value

- Hash value.

Type: [UInt64](#).

### Example

Query:

```
SELECT normalizedQueryHash('SELECT 1 AS `xyz`') != normalizedQueryHash('SELECT 1 AS `abc`') AS res;
```

Result:

```
res  
1 |
```

## normalizeUTF8NFC

Converts a string to [NFC normalized form](#), assuming the string contains a set of bytes that make up a UTF-8 encoded text.

### Syntax

```
normalizeUTF8NFC(words)
```

### Arguments

- words — Input string that contains UTF-8 encoded text. [String](#).

### Returned value

- String transformed to NFC normalization form.

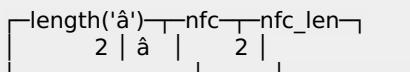
Type: [String](#).

### Example

Query:

```
SELECT length('â'), normalizeUTF8NFC('â') AS nfc, length(nfc) AS nfc_len;
```

Result:



## normalizeUTF8NFD

Converts a string to [NFD normalized form](#), assuming the string contains a set of bytes that make up a UTF-8 encoded text.

### Syntax

```
normalizeUTF8NFD(words)
```

### Arguments

- words — Input string that contains UTF-8 encoded text. [String](#).

### Returned value

- String transformed to NFD normalization form.

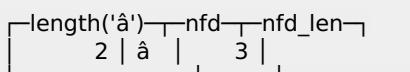
Type: [String](#).

### Example

Query:

```
SELECT length('â'), normalizeUTF8NFD('â') AS nfd, length(nfd) AS nfd_len;
```

Result:



## normalizeUTF8NFKC

Converts a string to [NFKC normalized form](#), assuming the string contains a set of bytes that make up a UTF-8 encoded text.

### Syntax

```
normalizeUTF8NFKC(words)
```

### Arguments

- words — Input string that contains UTF-8 encoded text. [String](#).

### Returned value

- String transformed to NFKC normalization form.

Type: [String](#).

## Example

Query:

```
SELECT length('â'), normalizeUTF8NFKC('â') AS nfkc, length(nfkc) AS nfkc_len;
```

Result:

length('â')	nfkc	nfkc_len
2   â		2

## normalizeUTF8NFKD

Converts a string to [NFKD normalized form](#), assuming the string contains a set of bytes that make up a UTF-8 encoded text.

## Syntax

```
normalizeUTF8NFKD(words)
```

## Arguments

- words — Input string that contains UTF-8 encoded text. [String](#).

## Returned value

- String transformed to NFKD normalization form.

Type: [String](#).

## Example

Query:

```
SELECT length('â'), normalizeUTF8NFKD('â') AS nfkd, length(nfkd) AS nfkd_len;
```

Result:

length('â')	nfkd	nfkd_len
2   â		3

## encodeXMLComponent

Escapes characters to place string into XML text node or attribute.

The following five XML predefined entities will be replaced: <, &, >, ", '.

## Syntax

```
encodeXMLComponent(x)
```

## Arguments

- `x` — The sequence of characters. [String](#).

## Returned value

- The sequence of characters with escape characters.

Type: [String](#).

## Example

Query:

```
SELECT encodeXMLComponent('Hello, "world"!');  
SELECT encodeXMLComponent('<123>');  
SELECT encodeXMLComponent('&clickhouse');  
SELECT encodeXMLComponent('\foo\');
```

Result:

```
Hello, "world"!<123>&clickhouse'&apos;foo&apos;
```

# decodeXMLComponent

Replaces XML predefined entities with characters. Predefined entities are `"`, `&`, `'`, `<`, `>`. This function also replaces numeric character references with Unicode characters. Both decimal (like `✓`) and hexadecimal (`✓`) forms are supported.

## Syntax

```
decodeXMLComponent(x)
```

## Arguments

- `x` — A sequence of characters. [String](#).

## Returned value

- The sequence of characters after replacement.

Type: [String](#).

## Example

Query:

```
SELECT decodeXMLComponent('foo');  
SELECT decodeXMLComponent('< >');
```

Result:

```
'foo'  
< >
```

## See Also

- List of XML and HTML character entity references

## extractTextFromHTML

A function to extract text from HTML or XHTML.

It does not necessarily 100% conform to any of the HTML, XML or XHTML standards, but the implementation is reasonably accurate and it is fast. The rules are the following:

1. Comments are skipped. Example: `<!-- test -->`. Comment must end with `-->`. Nested comments are not possible.  
Note: constructions like `<!-->` and `<!---->` are not valid comments in HTML but they are skipped by other rules.
2. CDATA is pasted verbatim. Note: CDATA is XML/XHTML specific. But it is processed for "best-effort" approach.
3. `script` and `style` elements are removed with all their content. Note: it is assumed that closing tag cannot appear inside content. For example, in JS string literal has to be escaped like `"<\script>"`.  
Note: comments and CDATA are possible inside `script` or `style` - then closing tags are not searched inside CDATA. Example: `<script><![CDATA[</script>]]></script>`. But they are still searched inside comments. Sometimes it becomes complicated: `<script>var x = "<!--"; </script> var y = "-->"; alert(x + y); </script>`  
Note: `script` and `style` can be the names of XML namespaces - then they are not treated like usual `script` or `style` elements. Example: `<script:a>Hello</script:a>`.  
Note: whitespaces are possible after closing tag name: `</script >` but not before: `< / script>`.
4. Other tags or tag-like elements are skipped without inner content. Example: `<a>.</a>`  
Note: it is expected that this HTML is illegal: `<a test=">"></a>`  
Note: it also skips something like tags: `<>`, `<!---->`, etc.  
Note: tag without end is skipped to the end of input: `<hello`
5. HTML and XML entities are not decoded. They must be processed by separate function.
6. Whitespace in the text are collapsed or inserted by specific rules.
  - Whitespace at the beginning and at the end are removed.
  - Consecutive whitespace are collapsed.
  - But if the text is separated by other elements and there is no whitespace, it is inserted.
  - It may cause unnatural examples: `Hello<b>world</b>`, `Hello<!-- -->world` - there is no whitespace in HTML, but the function inserts it. Also consider: `Hello<p>world</p>`, `Hello<br>world`. This behavior is reasonable for data analysis, e.g. to convert HTML to a bag of words.
7. Also note that correct handling of whitespace requires the support of `<pre></pre>` and CSS `display` and `white-space` properties.

### Syntax

```
extractTextFromHTML(x)
```

### Arguments

- `x` — input text. **String**.

### Returned value

- Extracted text.

Type: **String**.

### Example

The first example contains several tags and a comment and also shows whitespace processing.

The second example shows `CDATA` and `script` tag processing.

In the third example text is extracted from the full HTML response received by the `url` function.

Query:

```
SELECT extractTextFromHTML(' <p> A text <i>with</i><b>tags</b>. <!-- comments --> </p> ');
SELECT extractTextFromHTML('<![CDATA[The content within <b>CDATA</b>]]> <script>alert("Script");</script>');
SELECT extractTextFromHTML(html) FROM url('http://www.donothingfor2minutes.com/', RawBLOB, 'html String');
```

Result:

```
A text with tags .
The content within <b>CDATA</b>
Do Nothing for 2 Minutes 2:00 &nbsp;
```

## Functions for Searching in Strings

The search is case-sensitive by default in all these functions. There are separate variants for case insensitive search.

### Note

Functions for **replacing** and **other manipulations with strings** are described separately.

## position(haystack, needle), locate(haystack, needle)

Searches for the substring `needle` in the string `haystack`.

Returns the position (in bytes) of the found substring in the string, starting from 1.

For a case-insensitive search, use the function `positionCaseInsensitive`.

### Syntax

```
position(haystack, needle[, start_pos])
```

```
position(needle IN haystack)
```

Alias: `locate(haystack, needle[, start_pos])`.

### Note

Syntax of `position(needle IN haystack)` provides SQL-compatibility, the function works the same way as to `position(haystack, needle)`.

### Arguments

- `haystack` — String, in which substring will be searched. **String**.
- `needle` — Substring to be searched. **String**.

- `start_pos` – Position of the first character in the string to start search. **UInt**. Optional.

## Returned values

- Starting position in bytes (counting from 1), if substring was found.
- 0, if the substring was not found.

Type: `Integer`.

## Examples

The phrase “Hello, world!” contains a set of bytes representing a single-byte encoded text. The function returns some expected result:

Query:

```
SELECT position('Hello, world!', '!');
```

Result:

```
position('Hello, world!', '!')  
      13 |
```

```
SELECT  
  position('Hello, world!', 'o', 1),  
  position('Hello, world!', 'o', 7)
```

```
position('Hello, world!', 'o', 1) — position('Hello, world!', 'o', 7)  
      5 |           9 |
```

The same phrase in Russian contains characters which can't be represented using a single byte. The function returns some unexpected result (use `positionUTF8` function for multi-byte encoded text):

Query:

```
SELECT position('Привет, мир!', '!');
```

Result:

```
position('Привет, мир!', '!')  
      21 |
```

## Examples for POSITION(needle IN haystack) syntax

Query:

```
SELECT 3 = position('c' IN 'abc');
```

Result:

```
equals(3, position('abc', 'c'))  
  1 |
```

Query:

```
SELECT 6 = position('/' IN s) FROM (SELECT 'Hello/World' AS s);
```

Result:

```
equals(6, position(s, '/'))  
  1 |
```

## positionCaseInsensitive

The same as `position` returns the position (in bytes) of the found substring in the string, starting from 1. Use the function for a case-insensitive search.

Works under the assumption that the string contains a set of bytes representing a single-byte encoded text. If this assumption is not met and a character can't be represented using a single byte, the function does not throw an exception and returns some unexpected result. If character can be represented using two bytes, it will use two bytes and so on.

### Syntax

```
positionCaseInsensitive(haystack, needle[, start_pos])
```

### Arguments

- `haystack` — String, in which substring will be searched. [String](#).
- `needle` — Substring to be searched. [String](#).
- `start_pos` — Optional parameter, position of the first character in the string to start search. [UInt](#).

### Returned values

- Starting position in bytes (counting from 1), if substring was found.
- 0, if the substring was not found.

Type: [Integer](#).

### Example

Query:

```
SELECT positionCaseInsensitive('Hello, world!', 'hello');
```

Result:

```
positionCaseInsensitive('Hello, world!', 'hello')  
  1 |
```

# positionUTF8

Returns the position (in Unicode points) of the found substring in the string, starting from 1.

Works under the assumption that the string contains a set of bytes representing a UTF-8 encoded text. If this assumption is not met, the function does not throw an exception and returns some unexpected result. If character can be represented using two Unicode points, it will use two and so on.

For a case-insensitive search, use the function [positionCaseInsensitiveUTF8](#).

## Syntax

```
positionUTF8(haystack, needle[, start_pos])
```

## Arguments

- `haystack` — String, in which substring will be searched. [String](#).
- `needle` — Substring to be searched. [String](#).
- `start_pos` — Optional parameter, position of the first character in the string to start search. [UInt](#)

## Returned values

- Starting position in Unicode points (counting from 1), if substring was found.
- 0, if the substring was not found.

Type: [Integer](#).

## Examples

The phrase “Hello, world!” in Russian contains a set of Unicode points representing a single-point encoded text. The function returns some expected result:

Query:

```
SELECT positionUTF8('Привет, мир!', '!');
```

Result:

```
positionUTF8('Привет, мир!', '!')—  
12 |
```

The phrase “Salut, étudiante!”, where character é can be represented using a one point (U+00E9) or two points (U+0065U+0301) the function can be returned some unexpected result:

Query for the letter é, which is represented one Unicode point U+00E9:

```
SELECT positionUTF8('Salut, étudiante!', '!');
```

Result:

```
positionUTF8('Salut, étudiante!', '!')—  
17 |
```

Query for the letter é, which is represented two Unicode points U+0065U+0301:

```
SELECT positionUTF8('Salut, étudiante!', '!');
```

Result:

```
positionUTF8('Salut, étudiante!', '!')  
18 |
```

## positionCaseInsensitiveUTF8

The same as [positionUTF8](#), but is case-insensitive. Returns the position (in Unicode points) of the found substring in the string, starting from 1.

Works under the assumption that the string contains a set of bytes representing a UTF-8 encoded text. If this assumption is not met, the function does not throw an exception and returns some unexpected result. If character can be represented using two Unicode points, it will use two and so on.

### Syntax

```
positionCaseInsensitiveUTF8(haystack, needle[, start_pos])
```

### Arguments

- `haystack` — String, in which substring will be searched. [String](#).
- `needle` — Substring to be searched. [String](#).
- `start_pos` — Optional parameter, position of the first character in the string to start search. [UInt](#)

### Returned value

- Starting position in Unicode points (counting from 1), if substring was found.
- 0, if the substring was not found.

Type: Integer.

### Example

Query:

```
SELECT positionCaseInsensitiveUTF8('Привет, мир!', 'Мир');
```

Result:

```
positionCaseInsensitiveUTF8('Привет, мир!', 'Мир')  
9 |
```

## multiSearchAllPositions

The same as [position](#) but returns [Array](#) of positions (in bytes) of the found corresponding substrings in the string. Positions are indexed starting from 1.

The search is performed on sequences of bytes without respect to string encoding and collation.

- For case-insensitive ASCII search, use the function `multiSearchAllPositionsCaseInsensitive`.
- For search in UTF-8, use the function `multiSearchAllPositionsUTF8`.
- For case-insensitive UTF-8 search, use the function `multiSearchAllPositionsCaseInsensitiveUTF8`.

## Syntax

```
multiSearchAllPositions(haystack, [needle1, needle2, ..., needlen])
```

## Arguments

- `haystack` — String, in which substring will be searched. [String](#).
- `needle` — Substring to be searched. [String](#).

## Returned values

- Array of starting positions in bytes (counting from 1), if the corresponding substring was found and 0 if not found.

## Example

Query:

```
SELECT multiSearchAllPositions('Hello, World!', ['hello', '!', 'world']);
```

Result:

```
multiSearchAllPositions('Hello, World!', ['hello', '!', 'world'])—  
[0,13,0] |
```

## multiSearchAllPositionsUTF8

See `multiSearchAllPositions`.

## multiSearchFirstPosition(haystack, [needle<sub>1</sub>, needle<sub>2</sub>, ..., needle<sub>n</sub>])

The same as `position` but returns the leftmost offset of the string `haystack` that is matched to some of the needles.

For a case-insensitive search or/and in UTF-8 format use functions `multiSearchFirstPositionCaseInsensitive`, `multiSearchFirstPositionUTF8`, `multiSearchFirstPositionCaseInsensitiveUTF8`.

## multiSearchFirstIndex(haystack, [needle<sub>1</sub>, needle<sub>2</sub>, ..., needle<sub>n</sub>])

Returns the index  $i$  (starting from 1) of the leftmost found  $\text{needle}_i$  in the string `haystack` and 0 otherwise.

For a case-insensitive search or/and in UTF-8 format use functions `multiSearchFirstIndexCaseInsensitive`, `multiSearchFirstIndexUTF8`, `multiSearchFirstIndexCaseInsensitiveUTF8`.

## multiSearchAny(haystack, [needle<sub>1</sub>, needle<sub>2</sub>, ..., needle<sub>n</sub>])

Returns 1, if at least one string  $\text{needle}_i$  matches the string `haystack` and 0 otherwise.

For a case-insensitive search or/and in UTF-8 format use functions `multiSearchAnyCaseInsensitive`, `multiSearchAnyUTF8`, `multiSearchAnyCaseInsensitiveUTF8`.

## Note

In all `multiSearch*` functions the number of needles should be less than  $2^8$  because of implementation specification.

## match(haystack, pattern)

Checks whether the string matches the `pattern` regular expression. A `re2` regular expression. The [syntax](#) of the `re2` regular expressions is more limited than the syntax of the Perl regular expressions.

Returns 0 if it does not match, or 1 if it matches.

Note that the backslash symbol (\) is used for escaping in the regular expression. The same symbol is used for escaping in string literals. So in order to escape the symbol in a regular expression, you must write two backslashes (\) in a string literal.

The regular expression works with the string as if it is a set of bytes. The regular expression can't contain null bytes.

For patterns to search for substrings in a string, it is better to use `LIKE` or 'position', since they work much faster.

## multiMatchAny(haystack, [pattern<sub>1</sub>, pattern<sub>2</sub>, ..., pattern<sub>n</sub>])

The same as `match`, but returns 0 if none of the regular expressions are matched and 1 if any of the patterns matches. It uses [hyperscan](#) library. For patterns to search substrings in a string, it is better to use `multiSearchAny` since it works much faster.

## Note

The length of any of the haystack string must be less than  $2^{32}$  bytes otherwise the exception is thrown. This restriction takes place because of hyperscan API.

## multiMatchAnyIndex(haystack, [pattern<sub>1</sub>, pattern<sub>2</sub>, ..., pattern<sub>n</sub>])

The same as `multiMatchAny`, but returns any index that matches the haystack.

## multiMatchAllIndices(haystack, [pattern<sub>1</sub>, pattern<sub>2</sub>, ..., pattern<sub>n</sub>])

The same as `multiMatchAny`, but returns the array of all indices that match the haystack in any order.

## multiFuzzyMatchAny(haystack, distance, [pattern<sub>1</sub>, pattern<sub>2</sub>, ..., pattern<sub>n</sub>])

The same as `multiMatchAny`, but returns 1 if any pattern matches the haystack within a constant [edit distance](#). This function is also in an experimental mode and can be extremely slow. For more information see [hyperscan documentation](#).

## **multiFuzzyMatchAnyIndex(haystack, distance, [pattern<sub>1</sub>, pattern<sub>2</sub>, ..., pattern<sub>n</sub>])**

The same as `multiFuzzyMatchAny`, but returns any index that matches the haystack within a constant edit distance.

## **multiFuzzyMatchAllIndices(haystack, distance, [pattern<sub>1</sub>, pattern<sub>2</sub>, ..., pattern<sub>n</sub>])**

The same as `multiFuzzyMatchAny`, but returns the array of all indices in any order that match the haystack within a constant edit distance.

### **Note**

`multiFuzzyMatch*` functions do not support UTF-8 regular expressions, and such expressions are treated as bytes because of hyperscan restriction.

### **Note**

To turn off all functions that use hyperscan, use setting `SET allow_hyperscan = 0;`

## **extract(haystack, pattern)**

Extracts a fragment of a string using a regular expression. If 'haystack' does not match the 'pattern' regex, an empty string is returned. If the regex does not contain subpatterns, it takes the fragment that matches the entire regex. Otherwise, it takes the fragment that matches the first subpattern.

## **extractAll(haystack, pattern)**

Extracts all the fragments of a string using a regular expression. If 'haystack' does not match the 'pattern' regex, an empty string is returned. Returns an array of strings consisting of all matches to the regex. In general, the behavior is the same as the 'extract' function (it takes the first subpattern, or the entire expression if there isn't a subpattern).

## **extractAllGroupsHorizontal**

Matches all groups of the haystack string using the pattern regular expression. Returns an array of arrays, where the first array includes all fragments matching the first group, the second array - matching the second group, etc.

### **Note**

`extractAllGroupsHorizontal` function is slower than **extractAllGroupsVertical**.

## **Syntax**

```
extractAllGroupsHorizontal(haystack, pattern)
```

## **Arguments**

- `haystack` — Input string. Type: **String**.

- **pattern** — Regular expression with [re2 syntax](#). Must contain groups, each group enclosed in parentheses.  
If **pattern** contains no groups, an exception is thrown. Type: [String](#).

## Returned value

- Type: [Array](#).

If **haystack** does not match the **pattern** regex, an array of empty arrays is returned.

## Example

Query:

```
SELECT extractAllGroupsHorizontal('abc=111, def=222, ghi=333', '("[^"]+"|\w+)=("[^"]+"|\w+');
```

Result:

```
extractAllGroupsHorizontal('abc=111, def=222, ghi=333', '("[^"]+"|\w+)=("[^"]+"|\w+)') →  
[['abc','def','ghi'],['111','222','333']] |
```

## See Also

- [extractAllGroupsVertical](#)

# extractAllGroupsVertical

Matches all groups of the **haystack** string using the **pattern** regular expression. Returns an array of arrays, where each array includes matching fragments from every group. Fragments are grouped in order of appearance in the **haystack**.

## Syntax

```
extractAllGroupsVertical(haystack, pattern)
```

## Arguments

- **haystack** — Input string. Type: [String](#).
- **pattern** — Regular expression with [re2 syntax](#). Must contain groups, each group enclosed in parentheses.  
If **pattern** contains no groups, an exception is thrown. Type: [String](#).

## Returned value

- Type: [Array](#).

If **haystack** does not match the **pattern** regex, an empty array is returned.

## Example

Query:

```
SELECT extractAllGroupsVertical('abc=111, def=222, ghi=333', '("[^"]+"|\w+)=("[^"]+"|\w+');
```

Result:

```
extractAllGroupsVertical('abc=111, def=222, ghi=333', '(["^"]+"|\w+)=(["^"]+"|\w+')')  
[['abc','111'],['def','222'],['ghi','333']]
```

## See Also

- [extractAllGroupsHorizontal](#)

## like(haystack, pattern), haystack LIKE pattern operator

Checks whether a string matches a simple regular expression.

The regular expression can contain the metasymbols `%` and `_`.

`%` indicates any quantity of any bytes (including zero characters).

`_` indicates any one byte.

Use the backslash (`\`) for escaping metasymbols. See the note on escaping in the description of the 'match' function.

For regular expressions like `%needle%`, the code is more optimal and works as fast as the `position` function. For other regular expressions, the code is the same as for the 'match' function.

## notLike(haystack, pattern), haystack NOT LIKE pattern operator

The same thing as 'like', but negative.

## ilike

Case insensitive variant of [like](#) function. You can use `ILIKE` operator instead of the `ilike` function.

## Syntax

```
ilike(haystack, pattern)
```

## Arguments

- `haystack` — Input string. [String](#).
- `pattern` — If `pattern` does not contain percent signs or underscores, then the `pattern` only represents the string itself. An underscore (`_`) in `pattern` stands for (matches) any single character. A percent sign (`%`) matches any sequence of zero or more characters.

Some `pattern` examples:

```
'abc' ILIKE 'abc'    true  
'abc' ILIKE 'a%'    true  
'abc' ILIKE '_b_'   true  
'abc' ILIKE 'c'     false
```

## Returned values

- True, if the string matches `pattern`.
- False, if the string does not match `pattern`.

## Example

Input table:

id	name	days
1	January	31
2	February	29
3	March	31
4	April	30

Query:

```
SELECT * FROM Months WHERE ilike(name, '%j%');
```

Result:

id	name	days
1	January	31

## See Also

### ngramDistance(haystack, needle)

Calculates the 4-gram distance between haystack and needle: counts the symmetric difference between two multisets of 4-grams and normalizes it by the sum of their cardinalities. Returns float number from 0 to 1 – the closer to zero, the more strings are similar to each other. If the constant `needle` or `haystack` is more than 32Kb, throws an exception. If some of the non-constant `haystack` or `needle` strings are more than 32Kb, the distance is always one.

For case-insensitive search or/and in UTF-8 format use functions `ngramDistanceCaseInsensitive`, `ngramDistanceUTF8`, `ngramDistanceCaseInsensitiveUTF8`.

### ngramSearch(haystack, needle)

Same as `ngramDistance` but calculates the non-symmetric difference between `needle` and `haystack` – the number of n-grams from `needle` minus the common number of n-grams normalized by the number of `needle` n-grams. The closer to one, the more likely `needle` is in the `haystack`. Can be useful for fuzzy string search.

For case-insensitive search or/and in UTF-8 format use functions `ngramSearchCaseInsensitive`, `ngramSearchUTF8`, `ngramSearchCaseInsensitiveUTF8`.

## Note

For UTF-8 case we use 3-gram distance. All these are not perfectly fair n-gram distances. We use 2-byte hashes to hash n-grams and then calculate the (non-)symmetric difference between these hash tables – collisions may occur. With UTF-8 case-insensitive format we do not use fair `tolower` function – we zero the 5-th bit (starting from zero) of each codepoint byte and first bit of zeroth byte if bytes more than one – this works for Latin and mostly for all Cyrillic letters.

## countSubstrings

Returns the number of substring occurrences.

For a case-insensitive search, use `countSubstringsCaseInsensitive` or `countSubstringsCaseInsensitiveUTF8` functions.

## Syntax

```
countSubstrings(haystack, needle[, start_pos])
```

## Arguments

- `haystack` — The string to search in. [String](#).
- `needle` — The substring to search for. [String](#).
- `start_pos` – Position of the first character in the string to start search. Optional. [UInt](#).

## Returned values

- Number of occurrences.

Type: [UInt64](#).

## Examples

Query:

```
SELECT countSubstrings('foobar.com', '.');
```

Result:

```
└─countSubstrings('foobar.com', '.')─  
    1 |
```

Query:

```
SELECT countSubstrings('aaaa', 'aa');
```

Result:

```
└─countSubstrings('aaaa', 'aa')─  
    2 |
```

Query:

```
SELECT countSubstrings('abc__abc', 'abc', 4);
```

Result:

```
└─countSubstrings('abc__abc', 'abc', 4)─  
    1 |
```

# countSubstringsCaseInsensitive

Returns the number of substring occurrences case-insensitive.

## Syntax

```
countSubstringsCaseInsensitive(haystack, needle[, start_pos])
```

## Arguments

- `haystack` — The string to search in. [String](#).
- `needle` — The substring to search for. [String](#).
- `start_pos` — Position of the first character in the string to start search. Optional. [UInt](#).

## Returned values

- Number of occurrences.

Type: [UInt64](#).

## Examples

Query:

```
SELECT countSubstringsCaseInsensitive('aba', 'B');
```

Result:

```
countSubstringsCaseInsensitive('aba', 'B')  
1 |
```

Query:

```
SELECT countSubstringsCaseInsensitive('foobar.com', 'CoM');
```

Result:

```
countSubstringsCaseInsensitive('foobar.com', 'CoM')  
1 |
```

Query:

```
SELECT countSubstringsCaseInsensitive('abC__abC', 'aBc', 2);
```

Result:

```
countSubstringsCaseInsensitive('abC__abC', 'aBc', 2)  
1 |
```

## countSubstringsCaseInsensitiveUTF8

Returns the number of substring occurrences in [UTF-8](#) case-insensitive.

## Syntax

```
SELECT countSubstringsCaseInsensitiveUTF8(haystack, needle[, start_pos])
```

## Arguments

- `haystack` — The string to search in. [String](#).
- `needle` — The substring to search for. [String](#).
- `start_pos` — Position of the first character in the string to start search. Optional. [UInt](#).

## Returned values

- Number of occurrences.

Type: [UInt64](#).

## Examples

Query:

```
SELECT countSubstringsCaseInsensitiveUTF8('абв', 'А');
```

Result:

```
countSubstringsCaseInsensitiveUTF8('абв', 'А')  
1 |
```

Query:

```
SELECT countSubstringsCaseInsensitiveUTF8('аБв_Абв_абв', 'Абв');
```

Result:

```
countSubstringsCaseInsensitiveUTF8('аБв_Абв_абв', 'Абв')  
3 |
```

## countMatches(haystack, pattern)

Returns the number of regular expression matches for a pattern in a haystack.

## Syntax

```
countMatches(haystack, pattern)
```

## Arguments

- `haystack` — The string to search in. [String](#).
- `pattern` — The regular expression with [re2 syntax](#). [String](#).

## Returned value

- The number of matches.

Type: [UInt64](#).

## Examples

Query:

```
SELECT countMatches('foobar.com', 'o+');
```

Result:

```
countMatches('foobar.com', 'o+')—  
2 |—————
```

Query:

```
SELECT countMatches('aaaa', 'aa');
```

Result:

```
countMatches('aaaa', 'aa')——  
2 |—————
```

# Functions for Searching and Replacing in Strings

## Note

Functions for **searching** and **other manipulations with strings** are described separately.

## replaceOne(haystack, pattern, replacement)

Replaces the first occurrence, if it exists, of the ‘pattern’ substring in ‘haystack’ with the ‘replacement’ substring.

Hereafter, ‘pattern’ and ‘replacement’ must be constants.

## replaceAll(haystack, pattern, replacement), replace(haystack, pattern, replacement)

Replaces all occurrences of the ‘pattern’ substring in ‘haystack’ with the ‘replacement’ substring.

## replaceRegexpOne(haystack, pattern, replacement)

Replacement using the ‘pattern’ regular expression. A re2 regular expression.

Replaces only the first occurrence, if it exists.

A pattern can be specified as ‘replacement’. This pattern can include substitutions \0-\9.

The substitution \0 includes the entire regular expression. Substitutions \1-\9 correspond to the subpattern numbers. To use the \ character in a template, escape it using \\.

Also keep in mind that a string literal requires an extra escape.

Example 1. Converting the date to American format:

```

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

Example 2. Copying a string ten times:

```
SELECT replaceRegexpOne('Hello, World!', '.', '\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0') AS res
```

## replaceRegexpAll(haystack, pattern, replacement)

This does the same thing, but replaces all the occurrences. Example:

```
SELECT replaceRegexpAll('Hello, World!', '.', '\\0\\0') AS res
```

As an exception, if a regular expression worked on an empty substring, the replacement is not made more than once.

Example:

```
SELECT replaceRegexpAll('Hello, World!', '^', 'here: ') AS res
```

## regexpQuoteMeta(s)

The function adds a backslash before some predefined characters in the string.

Predefined characters: \0, \\, |, (, ), ^, \$, ., [ , ], ?, \*, +, {, :, -.

This implementation slightly differs from re2::RE2::QuoteMeta. It escapes zero byte as \0 instead of \x00 and it escapes only required characters.

For more information, see the link: [RE2](#)

# Conditional Functions

## if

Controls conditional branching. Unlike most systems, ClickHouse always evaluate both expressions `then` and `else`.

### Syntax

```
if(cond, then, else)
```

If the condition `cond` evaluates to a non-zero value, returns the result of the expression `then`, and the result of the expression `else`, if present, is skipped. If the `cond` is zero or `NULL`, then the result of the `then` expression is skipped and the result of the `else` expression, if present, is returned.

You can use the [short\\_circuit\\_function\\_evaluation](#) setting to calculate the `if` function according to a short scheme. If this setting is enabled, `then` expression is evaluated only on rows where `cond` is true, `else` expression – where `cond` is false. For example, an exception about division by zero is not thrown when executing the query `SELECT if(number = 0, 0, intDiv(42, number)) FROM numbers(10)`, because `intDiv(42, number)` will be evaluated only for numbers that doesn't satisfy condition `number = 0`.

### Arguments

- `cond` – The condition for evaluation that can be zero or not. The type is `UInt8`, `Nullable(UInt8)` or `NULL`.
- `then` – The expression to return if condition is met.
- `else` – The expression to return if condition is not met.

### Returned values

The function executes `then` and `else` expressions and returns its result, depending on whether the condition `cond` ended up being zero or not.

### Example

Query:

```
SELECT if(1, plus(2, 2), plus(2, 6));
```

Result:

```
plus(2, 2)─  
 4 |
```

Query:

```
SELECT if(0, plus(2, 2), plus(2, 6));
```

Result:

```
plus(2, 6)─  
 8 |
```

- `then` and `else` must have the lowest common type.

### Example:

Take this `LEFT_RIGHT` table:

```
SELECT *
FROM LEFT_RIGHT
```

left	right
NULL	4
1	3
2	2
3	1
4	NULL

The following query compares `left` and `right` values:

```
SELECT
  left,
  right,
  if(left < right, 'left is smaller than right', 'right is greater or equal than left') AS is_smaller
FROM LEFT_RIGHT
WHERE isNotNull(left) AND isNotNull(right)
```

left	right	is_smaller
1	3	left is smaller than right
2	2	right is greater or equal than left
3	1	right is greater or equal than left

Note: `NULL` values are not used in this example, check [NULL values in conditionals](#) section.

## Ternary Operator

It works same as `if` function.

Syntax: `cond ? then : else`

Returns `then` if the `cond` evaluates to be true (greater than zero), otherwise returns `else`.

- `cond` must be of type of `UInt8`, and `then` and `else` must have the lowest common type.
- `then` and `else` can be `NULL`

### See also

- [ifNotFinite](#).

## multilf

Allows you to write the `CASE` operator more compactly in the query.

### Syntax

```
multilf(cond_1, then_1, cond_2, then_2, ..., else)
```

You can use the `short_circuit_function_evaluation` setting to calculate the `multilf` function according to a short scheme. If this setting is enabled, `then_i` expression is evaluated only on rows where `((NOT cond_1) AND (NOT cond_2) AND ... AND (NOT cond_{i-1}) AND cond_i)` is true, `cond_i` will be evaluated only on rows where `((NOT`

`cond_1) AND (NOT cond_2) AND ... AND (NOT cond_{i-1})`) is true. For example, an exception about division by zero is not thrown when executing the query `SELECT multilf(number = 2, intDiv(1, number), number = 5) FROM numbers(10)`.

## Arguments

- `cond_N` — The condition for the function to return `then_N`.
- `then_N` — The result of the function when executed.
- `else` — The result of the function if none of the conditions is met.

The function accepts  $2N+1$  parameters.

## Returned values

The function returns one of the values `then_N` or `else`, depending on the conditions `cond_N`.

## Example

Again using `LEFT_RIGHT` table.

```
SELECT
    left,
    right,
    multilf(left < right, 'left is smaller', left > right, 'left is greater', left = right, 'Both equal', 'Null value') AS result
FROM LEFT_RIGHT
```

left	right	result
NULL	4	Null value
1	3	left is smaller
2	2	Both equal
3	1	left is greater
4	NULL	Null value

## Using Conditional Results Directly

Conditionals always result to 0, 1 or NULL. So you can use conditional results directly like this:

```
SELECT left < right AS is_small
FROM LEFT_RIGHT
```

is_small
NULL
1
0
0
NULL

## NULL Values in Conditionals

When NULL values are involved in conditionals, the result will also be NULL.

```
SELECT
    NULL < 1,
    2 < NULL,
    NULL < NULL,
    NULL = NULL
```

less(NULL, 1)	less(2, NULL)	less(NULL, NULL)	equals(NULL, NULL)
NULL	NULL	NULL	NULL

So you should construct your queries carefully if the types are Nullable.

The following example demonstrates this by failing to add equals condition to `multilf`.

```
SELECT
    left,
    right,
    multilf(left < right, 'left is smaller', left > right, 'right is smaller', 'Both equal') AS faulty_result
FROM LEFT_RIGHT
```

left	right	faulty_result
NULL	4	Both equal
1	3	left is smaller
2	2	Both equal
3	1	right is smaller
4	NULL	Both equal

## Functions for Working with Files

### file

Reads file as a String. The file content is not parsed, so any information is read as one string and placed into the specified column.

#### Syntax

```
file(path)
```

#### Arguments

- `path` — The relative path to the file from `user_files_path`. Path to file support following wildcards: `*`, `?`, `{abc,def}` and `{N..M}` where `N, M` — numbers, `'abc', 'def'` — strings.

#### Example

Inserting data from files `a.txt` and `b.txt` into a table as strings:

Query:

```
INSERT INTO table SELECT file('a.txt'), file('b.txt');
```

#### See Also

- [user\\_files\\_path](#)
- [file](#)

## Mathematical Functions

All the functions return a `Float64` number. The accuracy of the result is close to the maximum precision possible, but the result might not coincide with the machine representable number nearest to the corresponding real number.

### e()

Returns a `Float64` number that is close to the number e.

## **pi()**

Returns a Float64 number that is close to the number  $\pi$ .

## **exp(x)**

Accepts a numeric argument and returns a Float64 number close to the exponent of the argument.

## **log(x), ln(x)**

Accepts a numeric argument and returns a Float64 number close to the natural logarithm of the argument.

## **exp2(x)**

Accepts a numeric argument and returns a Float64 number close to 2 to the power of x.

## **log2(x)**

Accepts a numeric argument and returns a Float64 number close to the binary logarithm of the argument.

## **exp10(x)**

Accepts a numeric argument and returns a Float64 number close to 10 to the power of x.

## **log10(x)**

Accepts a numeric argument and returns a Float64 number close to the decimal logarithm of the argument.

## **sqrt(x)**

Accepts a numeric argument and returns a Float64 number close to the square root of the argument.

## **cbrt(x)**

Accepts a numeric argument and returns a Float64 number close to the cubic root of the argument.

## **erf(x)**

If 'x' is non-negative, then  $\text{erf}(x / \sigma\sqrt{2})$  is the probability that a random variable having a normal distribution with standard deviation ' $\sigma$ ' takes the value that is separated from the expected value by more than 'x'.

Example (three sigma rule):

```
SELECT erf(3 / sqrt(2));
```

```
erf(divide(3, sqrt(2)))—  
0.9973002039367398 |
```

## **erfc(x)**

Accepts a numeric argument and returns a Float64 number close to  $1 - \text{erf}(x)$ , but without loss of precision for large 'x' values.

## **lgamma(x)**

The logarithm of the gamma function.

## **tgamma(x)**

Gamma function.

## **sin(x)**

The sine.

## **cos(x)**

The cosine.

## **tan(x)**

The tangent.

## **asin(x)**

The arc sine.

## **acos(x)**

The arc cosine.

## **atan(x)**

The arc tangent.

## **pow(x, y), power(x, y)**

Takes two numeric arguments x and y. Returns a Float64 number close to x to the power of y.

## **intExp2**

Accepts a numeric argument and returns a UInt64 number close to 2 to the power of x.

## **intExp10**

Accepts a numeric argument and returns a UInt64 number close to 10 to the power of x.

## **cosh(x)**

Hyperbolic cosine.

### **Syntax**

```
cosh(x)
```

### **Arguments**

- $x$  — The angle, in radians. Values from the interval:  $-\infty < x < +\infty$ . **Float64**.

### **Returned value**

- Values from the interval:  $1 \leq \cosh(x) < +\infty$ .

Type: **Float64**.

### **Example**

Query:

```
SELECT cosh(0);
```

Result:

```
cosh(0)  
1 |
```

## acosh(x)

Inverse hyperbolic cosine.

### Syntax

```
acosh(x)
```

### Arguments

- x — Hyperbolic cosine of angle. Values from the interval:  $1 \leq x < +\infty$ . [Float64](#).

### Returned value

- The angle, in radians. Values from the interval:  $0 \leq \text{acosh}(x) < +\infty$ .

Type: [Float64](#).

### Example

Query:

```
SELECT acosh(1);
```

Result:

```
acosh(1)  
0 |
```

### See Also

- [cosh\(x\)](#)

## sinh(x)

Hyperbolic sine.

### Syntax

```
sinh(x)
```

### Arguments

- x — The angle, in radians. Values from the interval:  $-\infty < x < +\infty$ . [Float64](#).

### Returned value

- Values from the interval:  $-\infty < \sinh(x) < +\infty$ .

Type: [Float64](#).

## Example

Query:

```
SELECT sinh(0);
```

Result:

```
sinh(0)
0 |
```

## asinh(x)

[Inverse hyperbolic sine](#).

## Syntax

```
asinh(x)
```

## Arguments

- $x$  — Hyperbolic sine of angle. Values from the interval:  $-\infty < x < +\infty$ . [Float64](#).

## Returned value

- The angle, in radians. Values from the interval:  $-\infty < \operatorname{asinh}(x) < +\infty$ .

Type: [Float64](#).

## Example

Query:

```
SELECT asinh(0);
```

Result:

```
asinh(0)
0 |
```

## See Also

- [sinh\(x\)](#)

## atanh(x)

[Inverse hyperbolic tangent](#).

## Syntax

```
atanh(x)
```

## Arguments

- $x$  — Hyperbolic tangent of angle. Values from the interval:  $-1 < x < 1$ . **Float64**.

## Returned value

- The angle, in radians. Values from the interval:  $-\infty < \operatorname{atanh}(x) < +\infty$ .

Type: **Float64**.

## Example

Query:

```
SELECT atanh(0);
```

Result:

```
atanh(0)
0 |
```

## atan2(y, x)

The **function** calculates the angle in the Euclidean plane, given in radians, between the positive x axis and the ray to the point  $(x, y) \neq (0, 0)$ .

## Syntax

```
atan2(y, x)
```

## Arguments

- $y$  — y-coordinate of the point through which the ray passes. **Float64**.
- $x$  — x-coordinate of the point through which the ray passes. **Float64**.

## Returned value

- The angle  $\theta$  such that  $-\pi < \theta \leq \pi$ , in radians.

Type: **Float64**.

## Example

Query:

```
SELECT atan2(1, 1);
```

Result:

```
atan2(1, 1)
0.7853981633974483 |
```

## hypot(x, y)

Calculates the length of the hypotenuse of a right-angle triangle. The **function** avoids problems that occur when squaring very large or very small numbers.

## Syntax

```
hypot(x, y)
```

## Arguments

- `x` — The first cathetus of a right-angle triangle. **Float64**.
- `y` — The second cathetus of a right-angle triangle. **Float64**.

## Returned value

- The length of the hypotenuse of a right-angle triangle.

Type: **Float64**.

## Example

Query:

```
SELECT hypot(1, 1);
```

Result:

```
hypot(1, 1) |  
1.4142135623730951 |
```

## log1p(x)

Calculates  $\log(1+x)$ . The **function** `log1p(x)` is more accurate than `log(1+x)` for small values of `x`.

## Syntax

```
log1p(x)
```

## Arguments

- `x` — Values from the interval:  $-1 < x < +\infty$ . **Float64**.

## Returned value

- Values from the interval:  $-\infty < \log1p(x) < +\infty$ .

Type: **Float64**.

## Example

Query:

```
SELECT log1p(0);
```

Result:

```
log1p(0)
0 |
```

## See Also

- [log\(x\)](#)

## sign(x)

Returns the sign of a real number.

## Syntax

```
sign(x)
```

## Arguments

- $x$  — Values from  $-\infty$  to  $+\infty$ . Support all numeric types in ClickHouse.

## Returned value

- -1 for  $x < 0$
- 0 for  $x = 0$
- 1 for  $x > 0$

## Examples

Sign for the zero value:

```
SELECT sign(0);
```

Result:

```
sign(0)
0 |
```

Sign for the positive value:

```
SELECT sign(1);
```

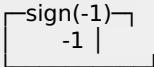
Result:

```
sign(1)
1 |
```

Sign for the negative value:

```
SELECT sign(-1);
```

Result:



# Rounding Functions

## `floor(x[, N])`

Returns the largest round number that is less than or equal to `x`. A round number is a multiple of  $1/10^N$ , or the nearest number of the appropriate data type if  $1 / 10^N$  isn't exact.

'`N`' is an integer constant, optional parameter. By default it is zero, which means to round to an integer. '`N`' may be negative.

Examples: `floor(123.45, 1) = 123.4`, `floor(123.45, -1) = 120`.

`x` is any numeric type. The result is a number of the same type.

For integer arguments, it makes sense to round with a negative `N` value (for non-negative `N`, the function does not do anything).

If rounding causes overflow (for example, `floor(-128, -1)`), an implementation-specific result is returned.

## `ceil(x[, N])`, `ceiling(x[, N])`

Returns the smallest round number that is greater than or equal to `x`. In every other way, it is the same as the `floor` function (see above).

## `trunc(x[, N])`, `truncate(x[, N])`

Returns the round number with largest absolute value that has an absolute value less than or equal to `x`'s. In every other way, it is the same as the 'floor' function (see above).

## `round(x[, N])`

Rounds a value to a specified number of decimal places.

The function returns the nearest number of the specified order. In case when given number has equal distance to surrounding numbers, the function uses banker's rounding for float number types and rounds away from zero for the other number types (Decimal).

```
round(expression [, decimal_places])
```

### Arguments

- `expression` — A number to be rounded. Can be any **expression** returning the numeric **data type**.
- `decimal-places` — An integer value.
  - If `decimal-places > 0` then the function rounds the value to the right of the decimal point.
  - If `decimal-places < 0` then the function rounds the value to the left of the decimal point.
  - If `decimal-places = 0` then the function rounds the value to integer. In this case the argument can be omitted.

### Returned value:

The rounded number of the same type as the input number.

## Examples

## Example of use with Float

```
SELECT number / 2 AS x, round(x) FROM system.numbers LIMIT 3
```

x	round(divide(number, 2))
0	0
0.5	0
1	1

## Example of use with Decimal

```
SELECT cast(number / 2 AS Decimal(10,4)) AS x, round(x) FROM system.numbers LIMIT 3
```

x	round(ROUND(divide(number, 2), 'Decimal(10, 4)'))
0.0000	0.0000
0.5000	1.0000
1.0000	1.0000

## Examples of rounding

Rounding to the nearest number.

```
round(3.2, 0) = 3  
round(4.1267, 2) = 4.13  
round(22,-1) = 20  
round(467,-2) = 500  
round(-467,-2) = -500
```

Banker's rounding.

```
round(3.5) = 4  
round(4.5) = 4  
round(3.55, 1) = 3.6  
round(3.65, 1) = 3.6
```

## See Also

- [roundBankers](#)

## roundBankers

Rounds a number to a specified decimal position.

- If the rounding number is halfway between two numbers, the function uses banker's rounding.

Banker's rounding is a method of rounding fractional numbers. When the rounding number is halfway between two numbers, it's rounded to the nearest even digit at the specified decimal position. For example: 3.5 rounds up to 4, 2.5 rounds down to 2.

It's the default rounding method for floating point numbers defined in [IEEE 754] ([https://en.wikipedia.org/wiki/IEEE\\_754#Roundings\\_to\\_nearest](https://en.wikipedia.org/wiki/IEEE_754#Roundings_to_nearest)). The [round](#rounding\_functions-round) function performs the same rounding for floating point numbers. The `roundBankers` function also rounds integers the same way, for example, `roundBankers(45, -1) = 40`.

- In other cases, the function rounds numbers to the nearest integer.

Using banker's rounding, you can reduce the effect that rounding numbers has on the results of summing or subtracting these numbers.

For example, sum numbers 1.5, 2.5, 3.5, 4.5 with different rounding:

- No rounding:  $1.5 + 2.5 + 3.5 + 4.5 = 12$ .
- Banker's rounding:  $2 + 2 + 4 + 4 = 12$ .
- Rounding to the nearest integer:  $2 + 3 + 4 + 5 = 14$ .

## Syntax

```
roundBankers(expression [, decimal_places])
```

## Arguments

- **expression** — A number to be rounded. Can be any **expression** returning the numeric **data type**.
- **decimal-places** — Decimal places. An integer number.
  - **decimal-places > 0** — The function rounds the number to the given position right of the decimal point.  
Example: `roundBankers(3.55, 1) = 3.6`.
  - **decimal-places < 0** — The function rounds the number to the given position left of the decimal point.  
Example: `roundBankers(24.55, -1) = 20`.
  - **decimal-places = 0** — The function rounds the number to an integer. In this case the argument can be omitted. Example: `roundBankers(2.5) = 2`.

## Returned value

A value rounded by the banker's rounding method.

## Examples

### Example of use

Query:

```
SELECT number / 2 AS x, roundBankers(x, 0) AS b fROM system.numbers limit 10
```

Result:

x	b
0	0
0.5	0
1	1
1.5	2
2	2
2.5	2
3	3
3.5	4
4	4
4.5	4

## Examples of Banker's rounding

```
roundBankers(0.4) = 0
roundBankers(-3.5) = -4
roundBankers(4.5) = 4
roundBankers(3.55, 1) = 3.6
roundBankers(3.65, 1) = 3.6
roundBankers(10.35, 1) = 10.4
roundBankers(10.755, 2) = 10.76
```

## See Also

- [round](#)

## roundToExp2(num)

Accepts a number. If the number is less than one, it returns 0. Otherwise, it rounds the number down to the nearest (whole non-negative) degree of two.

## roundDuration(num)

Accepts a number. If the number is less than one, it returns 0. Otherwise, it rounds the number down to numbers from the set: 1, 10, 30, 60, 120, 180, 240, 300, 600, 1200, 1800, 3600, 7200, 18000, 36000. This function is specific to Yandex.Metrica and used for implementing the report on session length.

## roundAge(num)

Accepts a number. If the number is less than 18, it returns 0. Otherwise, it rounds the number down to a number from the set: 18, 25, 35, 45, 55. This function is specific to Yandex.Metrica and used for implementing the report on user age.

## roundDown(num, arr)

Accepts a number and rounds it down to an element in the specified array. If the value is less than the lowest bound, the lowest bound is returned.

# Functions for maps

## map

Arranges key:value pairs into [Map\(key, value\)](#) data type.

### Syntax

```
map(key1, value1[, key2, value2, ...])
```

### Arguments

- **key** — The key part of the pair. [String](#) or [Integer](#).
- **value** — The value part of the pair. [String](#), [Integer](#) or [Array](#).

### Returned value

- Data structure as key:value pairs.

Type: [Map\(key, value\)](#).

### Examples

Query:

```
SELECT map('key1', number, 'key2', number * 2) FROM numbers(3);
```

Result:

```
map('key1', number, 'key2', multiply(number, 2))  
{'key1':0,'key2':0}  
{'key1':1,'key2':2}  
{'key1':2,'key2':4}
```

Query:

```
CREATE TABLE table_map (a Map(String, UInt64)) ENGINE = MergeTree() ORDER BY a;  
INSERT INTO table_map SELECT map('key1', number, 'key2', number * 2) FROM numbers(3);  
SELECT a['key2'] FROM table_map;
```

Result:

```
arrayElement(a, 'key2')  
0  
2  
4
```

## See Also

- [Map\(key, value\)](#) data type

## mapAdd

Collect all the keys and sum corresponding values.

### Syntax

```
mapAdd(arg1, arg2 [, ...])
```

### Arguments

Arguments are [maps](#) or [tuples](#) of two [arrays](#), where items in the first array represent keys, and the second array contains values for each key. All key arrays should have same type, and all value arrays should contain items which are promoted to the one type ([Int64](#), [UInt64](#) or [Float64](#)). The common promoted type is used as a type for the result array.

### Returned value

- Depending on the arguments returns one [map](#) or [tuple](#), where the first array contains the sorted keys and the second array contains values.

### Example

Query with a tuple:

```
SELECT mapAdd(([toUInt8(1), 2], [1, 1]), ([toUInt8(1), 2], [1, 1])) as res, toTypeName(res) as type;
```

Result:

```
res-----type
([1,2],[2,2]) | Tuple(Array(UInt8), Array(UInt64)) |
```

Query with Map type:

```
SELECT mapAdd(map(1,1), map(1,1));
```

Result:

```
mapAdd(map(1, 1), map(1, 1))-
{1:2}
```

## mapSubtract

Collect all the keys and subtract corresponding values.

### Syntax

```
mapSubtract(Tuple(Array, Array), Tuple(Array, Array) [, ...])
```

### Arguments

Arguments are **maps** or **tuples** of two **arrays**, where items in the first array represent keys, and the second array contains values for each key. All key arrays should have same type, and all value arrays should contain items which are promote to the one type (**Int64**, **UInt64** or **Float64**). The common promoted type is used as a type for the result array.

### Returned value

- Depending on the arguments returns one **map** or **tuple**, where the first array contains the sorted keys and the second array contains values.

### Example

Query with a tuple map:

```
SELECT mapSubtract(([toUInt8(1), 2], [toInt32(1), 1]), ([toUInt8(1), 2], [toInt32(2), 1])) as res, toTypeName(res) as
type;
```

Result:

```
res-----type
([1,2],[-1,0]) | Tuple(Array(UInt8), Array(Int64)) |
```

Query with Map type:

```
SELECT mapSubtract(map(1,1), map(1,1));
```

Result:

```
mapSubtract(map(1, 1), map(1, 1))  
{1:0}
```

## mapPopulateSeries

Fills missing keys in the maps (key and value array pair), where keys are integers. Also, it supports specifying the max key, which is used to extend the keys array.

### Syntax

```
mapPopulateSeries(keys, values[, max])  
mapPopulateSeries(map[, max])
```

Generates a map (a tuple with two arrays or a value of `Map` type, depending on the arguments), where keys are a series of numbers, from minimum to maximum keys (or `max` argument if it specified) taken from the map with a step size of one, and corresponding values. If the value is not specified for the key, then it uses the default value in the resulting map. For repeated keys, only the first value (in order of appearing) gets associated with the key.

For array arguments the number of elements in `keys` and `values` must be the same for each row.

### Arguments

Arguments are `maps` or two `arrays`, where the first array represent keys, and the second array contains values for the each key.

Mapped arrays:

- `keys` — Array of keys. [Array\(Int\)](#).
- `values` — Array of values. [Array\(Int\)](#).
- `max` — Maximum key value. Optional. [Int8](#), [Int16](#), [Int32](#), [Int64](#), [Int128](#), [Int256](#).

or

- `map` — Map with integer keys. [Map](#).

### Returned value

- Depending on the arguments returns a `map` or a `tuple` of two `arrays`: keys in sorted order, and values the corresponding keys.

### Example

Query with mapped arrays:

```
SELECT mapPopulateSeries([1,2,4], [11,22,44], 5) AS res, toTypeName(res) AS type;
```

Result:

```
res          type  
([1,2,3,4,5],[11,22,0,44,0]) | Tuple(Array(UInt8), Array(UInt8)) |
```

Query with `Map` type:

```
SELECT mapPopulateSeries(map(1, 10, 5, 20), 6);
```

Result:

```
mapPopulateSeries(map(1, 10, 5, 20), 6)─  
 {1:10,2:0,3:0,4:0,5:20,6:0} ─
```

## mapContains

Determines whether the `map` contains the `key` parameter.

### Syntax

```
mapContains(map, key)
```

### Parameters

- `map` — Map. [Map](#).
- `key` — Key. Type matches the type of keys of `map` parameter.

### Returned value

- 1 if `map` contains `key`, 0 if not.

Type: [UInt8](#).

### Example

Query:

```
CREATE TABLE test (a Map(String,String)) ENGINE = Memory;  
INSERT INTO test VALUES ( {'name':'eleven','age':'11'} ), ( {'number':'twelve','position':'6.0'} );  
SELECT mapContains(a, 'name') FROM test;
```

Result:

```
mapContains(a, 'name')─  
 1 |  
 0 ─
```

## mapKeys

Returns all keys from the `map` parameter.

Can be optimized by enabling the [optimize\\_functions\\_to\\_subcolumns](#) setting. With `optimize_functions_to_subcolumns = 1` the function reads only `keys` subcolumn instead of reading and processing the whole column data. The query `SELECT mapKeys(m) FROM table` transforms to `SELECT m.keys FROM table`.

### Syntax

```
mapKeys(map)
```

## Parameters

- `map` — Map. [Map](#).

## Returned value

- Array containing all keys from the `map`.

Type: [Array](#).

## Example

Query:

```
CREATE TABLE test (a Map(String,String)) ENGINE = Memory;
INSERT INTO test VALUES ({"name":'eleven','age':'11'}), ({"number":'twelve','position':'6.0'});
SELECT mapKeys(a) FROM test;
```

Result:

```
mapKeys(a)
['name','age']
  |
['number','position'] |
```

## mapValues

Returns all values from the `map` parameter.

Can be optimized by enabling the [optimize\\_functions\\_to\\_subcolumns](#) setting. With `optimize_functions_to_subcolumns = 1` the function reads only [values](#) subcolumn instead of reading and processing the whole column data. The query `SELECT mapValues(m) FROM table` transforms to `SELECT m.values FROM table`.

## Syntax

```
mapKeys(map)
```

## Parameters

- `map` — Map. [Map](#).

## Returned value

- Array containing all the values from `map`.

Type: [Array](#).

## Example

Query:

```
CREATE TABLE test (a Map(String,String)) ENGINE = Memory;
INSERT INTO test VALUES ({"name":'eleven','age':'11'}), ({"number":'twelve','position':'6.0'});
SELECT mapValues(a) FROM test;
```

Result:

```
mapValues(a)
['eleven','11']
['twelve','6.0']
```

## mapContainsKeyLike

### Syntax

```
mapContainsKeyLike(map, pattern)
```

### Parameters

- `map` — Map. [Map](#).
- `pattern` - String pattern to match.

### Returned value

- 1 if `map` contains key like specified pattern, 0 if not.

### Example

Query:

```
CREATE TABLE test (a Map(String,String)) ENGINE = Memory;
INSERT INTO test VALUES ( {'abc':'abc','def':'def'} ), ( {'hij':'hij','klm':'klm'} );
SELECT mapContainsKeyLike(a, 'a%') FROM test;
```

Result:

```
mapContainsKeyLike(a, 'a%')
1
0
```

## mapExtractKeyLike

### Syntax

```
mapExtractKeyLike(map, pattern)
```

### Parameters

- `map` — Map. [Map](#).
- `pattern` - String pattern to match.

### Returned value

- A map contained elements the key of which matchs the specified pattern. If there are no elements matched the pattern, it will return an empty map.

### Example

Query:

```
CREATE TABLE test (a Map(String,String)) ENGINE = Memory;  
INSERT INTO test VALUES ( {'abc':'abc','def':'def'} ), ( {'hij':'hij','klm':'klm'} );  
SELECT mapExtractKeyLike(a, 'a%') FROM test;
```

Result:

```
mapExtractKeyLike(a, 'a%')  
{'abc':'abc'}  
{}
```

## Functions for Splitting and Merging Strings and Arrays

### splitByChar(separator, s)

Splits a string into substrings separated by a specified character. It uses a constant string `separator` which consists of exactly one character.

Returns an array of selected substrings. Empty substrings may be selected if the separator occurs at the beginning or end of the string, or if there are multiple consecutive separators.

#### Syntax

```
splitByChar(separator, s)
```

#### Arguments

- `separator` — The separator which should contain exactly one character. [String](#).
- `s` — The string to split. [String](#).

#### Returned value(s)

Returns an array of selected substrings. Empty substrings may be selected when:

- A separator occurs at the beginning or end of the string;
- There are multiple consecutive separators;
- The original string `s` is empty.

Type: [Array\(String\)](#).

#### Example

```
SELECT splitByChar(',', '1,2,3,abcde');
```

```
splitByChar(',', '1,2,3,abcde')  
['1','2','3','abcde']
```

### splitByString(separator, s)

Splits a string into substrings separated by a string. It uses a constant string `separator` of multiple characters as the separator. If the string `separator` is empty, it will split the string `s` into an array of single characters.

## Syntax

```
splitByString(separator, s)
```

## Arguments

- `separator` — The separator. [String](#).
- `s` — The string to split. [String](#).

## Returned value(s)

Returns an array of selected substrings. Empty substrings may be selected when:

Type: [Array\(String\)](#).

- A non-empty separator occurs at the beginning or end of the string;
- There are multiple consecutive non-empty separators;
- The original string `s` is empty while the separator is not empty.

## Example

```
SELECT splitByString(' ', '1, 2 3, 4,5, abcde');
```

```
splitByString(' ', '1, 2 3, 4,5, abcde') └  
['1','2 3','4,5','abcde'] ┌
```

```
SELECT splitByString('', 'abcde');
```

```
splitByString('', 'abcde') └  
['a','b','c','d','e'] ┌
```

## splitByRegexp(regexp, s)

Splits a string into substrings separated by a regular expression. It uses a regular expression string `regexp` as the separator. If the `regexp` is empty, it will split the string `s` into an array of single characters. If no match is found for this regular expression, the string `s` won't be split.

## Syntax

```
splitByRegexp(regexp, s)
```

## Arguments

- `regexp` — Regular expression. Constant. [String](#) or [FixedString](#).
- `s` — The string to split. [String](#).

## Returned value(s)

Returns an array of selected substrings. Empty substrings may be selected when:

- A non-empty regular expression match occurs at the beginning or end of the string;
- There are multiple consecutive non-empty regular expression matches;
- The original string `s` is empty while the regular expression is not empty.

Type: `Array(String)`.

## Example

Query:

```
SELECT splitByRegexp('\\d+', 'a12bc23de345f');
```

Result:

```
splitByRegexp('\\d+', 'a12bc23de345f')  
['a','bc','de','f']
```

Query:

```
SELECT splitByRegexp('', 'abcde');
```

Result:

```
splitByRegexp('', 'abcde')  
['a','b','c','d','e']
```

## splitByWhitespace(s)

Splits a string into substrings separated by whitespace characters.

Returns an array of selected substrings.

## Syntax

```
splitByWhitespace(s)
```

## Arguments

- `s` — The string to split. `String`.

## Returned value(s)

Returns an array of selected substrings.

Type: `Array(String)`.

## Example

```
SELECT splitByWhitespace(' 1! a, b. ');
```

```
splitByWhitespace(' 1! a, b. ')
```

```
['1!', 'a', 'b. ']
```

## splitByNonAlpha(s)

Splits a string into substrings separated by whitespace and punctuation characters.

Returns an array of selected substrings.

### Syntax

```
splitByNonAlpha(s)
```

### Arguments

- s — The string to split. [String](#).

### Returned value(s)

Returns an array of selected substrings.

Type: [Array\(String\)](#).

### Example

```
SELECT splitByNonAlpha(' 1! a, b.');
```

```
splitByNonAlpha(' 1! a, b. ')
```

```
['1!', 'a', 'b. ']
```

## arrayStringConcat(arr[, separator])

Concatenates the strings (values of type String or Nullable(String)) listed in the array with the separator. 'separator' is an optional parameter: a constant string, set to an empty string by default.

Returns the string.

## alphaTokens(s)

Selects substrings of consecutive bytes from the ranges a-z and A-Z. Returns an array of substrings.

### Example

```
SELECT alphaTokens('abca1abc');
```

```
alphaTokens('abca1abc')
```

```
['abca', 'abc']
```

## extractAllGroups(text, regexp)

Extracts all groups from non-overlapping substrings matched by a regular expression.

### Syntax

```
extractAllGroups(text, regexp)
```

## Arguments

- `text` — **String** or **FixedString**.
- `regexp` — Regular expression. Constant. **String** or **FixedString**.

## Returned values

- If the function finds at least one matching group, it returns `Array(Array(String))` column, clustered by `group_id` (1 to N, where N is number of capturing groups in `regexp`).
- If there is no matching group, returns an empty array.

Type: **Array**.

## Example

Query:

```
SELECT extractAllGroups('abc=123, 8="hkl"', '("[^"]+"|\w+)=(["^"]+"|\w+');
```

Result:

```
extractAllGroups('abc=123, 8="hkl"', '("[^"]+"|\w+)=(["^"]+"|\w+')  
[['abc','123'],['8',"hkl"]]
```

# ngrams

Splits the UTF-8 string into n-grams of `ngramsize` symbols.

## Syntax

```
ngrams(string, ngramsize)
```

## Arguments

- `string` — String. **String** or **FixedString**.
- `ngramsize` — The size of an n-gram. **UInt**.

## Returned values

- Array with n-grams.

Type: **Array(FixedString)**.

## Example

Query:

```
SELECT ngrams('ClickHouse', 3);
```

Result:

```
ngrams('ClickHouse', 3)
['Cli','lic','ick','ckH','kHo','Hou','ous','use'] |
```

## tokens

Splits a string into tokens using non-alphanumeric ASCII characters as separators.

### Arguments

- `input_string` — Any set of bytes represented as the [String](#) data type object.

### Returned value

- The resulting array of tokens from input string.

Type: [Array](#).

### Example

Query:

```
SELECT tokens('test1,;\\ test2,;\\ test3,;\\  test4') AS tokens;
```

Result:

```
tokens
['test1','test2','test3','test4'] |
```

## Bit Functions

Bit functions work for any pair of types from `UInt8`, `UInt16`, `UInt32`, `UInt64`, `Int8`, `Int16`, `Int32`, `Int64`, `Float32`, or `Float64`.

The result type is an integer with bits equal to the maximum bits of its arguments. If at least one of the arguments is signed, the result is a signed number. If an argument is a floating-point number, it is cast to `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`

Takes any integer and converts it into **binary form**, returns the value of a bit at specified position. The countdown starts from 0 from the right to the left.

## Syntax

```
SELECT bitTest(number, index)
```

## Arguments

- **number** – Integer number.
- **index** – Position of bit.

## Returned values

Returns a value of bit at specified position.

Type: UInt8.

## Example

For example, the number 43 in base-2 (binary) numeral system is 101011.

Query:

```
SELECT bitTest(43, 1);
```

Result:

```
bitTest(43, 1) └  
      1 |
```

Another example:

Query:

```
SELECT bitTest(43, 2);
```

Result:

```
bitTest(43, 2) └  
      0 |
```

## bitTestAll

Returns result of **logical conjunction** (AND operator) of all bits at given positions. The countdown starts from 0 from the right to the left.

The conjunction for bitwise operations:

0 AND 0 = 0

0 AND 1 = 0

1 AND 0 = 0

1 AND 1 = 1

## Syntax

```
SELECT bitTestAll(number, index1, index2, index3, index4, ...)
```

## Arguments

- `number` – Integer number.
- `index1, index2, index3, index4` – Positions of bit. For example, for set of positions (`index1, index2, index3, index4`) is true if and only if all of its positions are true (`index1  $\wedge$  index2,  $\wedge$  index3  $\wedge$  index4`).

## Returned values

Returns result of logical conjunction.

Type: `UInt8`.

## Example

For example, the number 43 in base-2 (binary) numeral system is 101011.

Query:

```
SELECT bitTestAll(43, 0, 1, 3, 5);
```

Result:

```
bitTestAll(43, 0, 1, 3, 5)  
1 |
```

Another example:

Query:

```
SELECT bitTestAll(43, 0, 1, 3, 5, 2);
```

Result:

```
bitTestAll(43, 0, 1, 3, 5, 2)  
0 |
```

## bitTestAny

Returns result of **logical disjunction** (OR operator) of all bits at given positions. The countdown starts from 0 from the right to the left.

The disjunction for bitwise operations:

0 OR 0 = 0

0 OR 1 = 1

1 OR 0 = 1

1 OR 1 = 1

## Syntax

```
SELECT bitTestAny(number, index1, index2, index3, index4, ...)
```

## Arguments

- `number` – Integer number.
- `index1, index2, index3, index4` – Positions of bit.

## Returned values

Returns result of logical disjunction.

Type: `UInt8`.

## Example

For example, the number 43 in base-2 (binary) numeral system is 101011.

Query:

```
SELECT bitTestAny(43, 0, 2);
```

Result:

```
bitTestAny(43, 0, 2)
  1 |
```

Another example:

Query:

```
SELECT bitTestAny(43, 4, 2);
```

Result:

```
bitTestAny(43, 4, 2)
  0 |
```

## bitCount

Calculates the number of bits set to one in the binary representation of a number.

## Syntax

```
bitCount(x)
```

## Arguments

- `x` — **Integer** or **floating-point** number. The function uses the value representation in memory. It allows supporting floating-point numbers.

## Returned value

- Number of bits set to one in the input number.

The function does not convert input value to a larger type ([sign extension](#)). So, for example,

`bitCount(toUInt8(-1)) = 8.`

Type: [UInt8](#).

## Example

Take for example the number 333. Its binary representation: 0000000101001101.

Query:

```
SELECT bitCount(333);
```

Result:

```
bitCount(333)─  
      5 |
```

## bitHammingDistance

Returns the [Hamming Distance](#) between the bit representations of two integer values. Can be used with [SimHash](#) functions for detection of semi-duplicate strings. The smaller is the distance, the more likely those strings are the same.

## Syntax

```
bitHammingDistance(int1, int2)
```

## Arguments

- `int1` — First integer value. [Int64](#).
- `int2` — Second integer value. [Int64](#).

## Returned value

- The Hamming distance.

Type: [UInt8](#).

## Examples

Query:

```
SELECT bitHammingDistance(111, 121);
```

Result:

```
bitHammingDistance(111, 121)─  
      3 |
```

With [SimHash](#):

```
SELECT bitHammingDistance(ngramSimHash('cat ate rat'), ngramSimHash('rat ate cat'));
```

Result:

```
bitHammingDistance(ngramSimHash('cat ate rat'), ngramSimHash('rat ate cat'))—  
5 |
```

## Bitmap Functions

Bitmap functions work for two bitmap Object value calculation, it is to return new bitmap or cardinality while using formula calculation, such as and, or, xor, and not, etc.

There are 2 kinds of construction methods for Bitmap Object. One is to be constructed by aggregation function groupBitmap with -State, the other is to be constructed by Array Object. It is also to convert Bitmap Object to Array Object.

RoaringBitmap is wrapped into a data structure while actual storage of Bitmap objects. When the cardinality is less than or equal to 32, it uses Set object. When the cardinality is greater than 32, it uses RoaringBitmap object. That is why storage of low cardinality set is faster.

For more information on RoaringBitmap, see: [CRoaring](#).

### bitmapBuild

Build a bitmap from unsigned integer array.

```
bitmapBuild(array)
```

#### Arguments

- array – Unsigned integer array.

#### Example

```
SELECT bitmapBuild([1, 2, 3, 4, 5]) AS res, toTypeName(res);
```

```
res— toTypeName(bitmapBuild([1, 2, 3, 4, 5]))—  
| AggregateFunction(groupBitmap, UInt8) |
```

### bitmapToArray

Convert bitmap to integer array.

```
bitmapToArray(bitmap)
```

#### Arguments

- bitmap – Bitmap object.

#### Example

```
SELECT bitmapToArray(bitmapBuild([1, 2, 3, 4, 5])) AS res;
```

```
res  
[1,2,3,4,5] |
```

## bitmapSubsetInRange

Return subset in specified range (not include the range\_end).

```
bitmapSubsetInRange(bitmap, range_start, range_end)
```

### Arguments

- `bitmap` – [Bitmap object](#).
- `range_start` – Range start point. Type: [UInt32](#).
- `range_end` – Range end point (excluded). Type: [UInt32](#).

### Example

```
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

Creates a subset of bitmap with n elements taken between `range_start` and `cardinality_limit`.

### Syntax

```
bitmapSubsetLimit(bitmap, range_start, cardinality_limit)
```

### Arguments

- `bitmap` – [Bitmap object](#).
- `range_start` – The subset starting point. Type: [UInt32](#).
- `cardinality_limit` – The subset cardinality upper limit. Type: [UInt32](#).

### Returned value

The subset.

Type: [Bitmap object](#).

### Example

Query:

```
SELECT
bitmapToArray(bitmapSubsetLimit(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;
```

Result:

```
res
[30,31,32,33,100,200,500] |
```

## subBitmap

Returns the bitmap elements, starting from the `offset` position. The number of returned elements is limited by the `cardinality_limit` parameter. Analog of the `substring` string function, but for bitmap.

### Syntax

```
subBitmap(bitmap, offset, cardinality_limit)
```

### Arguments

- `bitmap` – The bitmap. Type: [Bitmap object](#).
- `offset` – The position of the first element of the subset. Type: [UInt32](#).
- `cardinality_limit` – The maximum number of elements in the subset. Type: [UInt32](#).

### Returned value

The subset.

Type: [Bitmap object](#).

### Example

Query:

```
SELECT
bitmapToArray(subBitmap(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(10), toUInt32(10))) AS res;
```

Result:

```
res
[10,11,12,13,14,15,16,17,18,19] |
```

## bitmapContains

Checks whether the bitmap contains an element.

```
bitmapContains(haystack, needle)
```

### Arguments

- `haystack` – [Bitmap object](#), where the function searches.

- **needle** – Value that the function searches. Type: [UInt32](#).

## Returned values

- 0 — If haystack does not contain needle.
- 1 — If haystack contains needle.

Type: [UInt8](#).

## Example

```
SELECT bitmapContains(bitmapBuild([1,5,7,9]), toUInt32(9)) AS res;
```

```
res  
1 |
```

## bitmapHasAny

Checks whether two bitmaps have intersection by some elements.

```
bitmapHasAny(bitmap1, bitmap2)
```

If you are sure that `bitmap2` contains strictly one element, consider using the [bitmapContains](#) function. It works more efficiently.

## Arguments

- `bitmap*` – Bitmap object.

## Return values

- 1, if `bitmap1` and `bitmap2` have one similar element at least.
- 0, otherwise.

## Example

```
SELECT bitmapHasAny(bitmapBuild([1,2,3]),bitmapBuild([3,4,5])) AS res;
```

```
res  
1 |
```

## bitmapHasAll

Analogous to `hasAll(array, array)` returns 1 if the first bitmap contains all the elements of the second one, 0 otherwise.

If the second argument is an empty bitmap then returns 1.

```
bitmapHasAll(bitmap,bitmap)
```

## Arguments

- `bitmap` – Bitmap object.

## Example

```
SELECT bitmapHasAll(bitmapBuild([1,2,3]),bitmapBuild([3,4,5])) AS res;
```

```
res  
0 |
```

## bitmapCardinality

Retrun bitmap cardinality of type UInt64.

```
bitmapCardinality(bitmap)
```

## Arguments

- `bitmap` – Bitmap object.

## Example

```
SELECT bitmapCardinality(bitmapBuild([1, 2, 3, 4, 5])) AS res;
```

```
res  
5 |
```

## bitmapMin

Retrun the smallest value of type UInt64 in the set, `UINT32_MAX` if the set is empty.

```
bitmapMin(bitmap)
```

## Arguments

- `bitmap` – Bitmap object.

## Example

```
SELECT bitmapMin(bitmapBuild([1, 2, 3, 4, 5])) AS res;
```

```
res  
1 |
```

## bitmapMax

Retrun the greatest value of type UInt64 in the set, 0 if the set is empty.

```
bitmapMax(bitmap)
```

## Arguments

- `bitmap` – Bitmap object.

## Example

```
SELECT bitmapMax(bitmapBuild([1, 2, 3, 4, 5])) AS res;
```

```
res  
5 |
```

## bitmapTransform

Transform an array of values in a bitmap to another array of values, the result is a new bitmap.

```
bitmapTransform(bitmap, from_array, to_array)
```

## Arguments

- `bitmap` – Bitmap object.
- `from_array` – UInt32 array. For idx in range [0, `from_array.size()`), if `bitmap` contains `from_array[idx]`, then replace it with `to_array[idx]`. Note that the result depends on array ordering if there are common elements between `from_array` and `to_array`.
- `to_array` – UInt32 array, its size shall be the same to `from_array`.

## Example

```
SELECT bitmapToArray(bitmapTransform(bitmapBuild([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]), cast([5,999,2] as Array(UInt32)),  
cast([2,888,20] as Array(UInt32)))) AS res;
```

```
res  
[1,3,4,6,7,8,9,10,20] |
```

## bitmapAnd

Two bitmap and calculation, the result is a new bitmap.

```
bitmapAnd(bitmap,bitmap)
```

## Arguments

- `bitmap` – Bitmap object.

## Example

```
SELECT bitmapToArray(bitmapAnd(bitmapBuild([1,2,3]),bitmapBuild([3,4,5]))) AS res;
```

```
res  
[3] |
```

## bitmapOr

Two bitmap or calculation, the result is a new bitmap.

```
bitmapOr(bitmap,bitmap)
```

### Arguments

- `bitmap` – Bitmap object.

### Example

```
SELECT bitmapToArray(bitmapOr(bitmapBuild([1,2,3]),bitmapBuild([3,4,5]))) AS res;
```

```
res  
[1,2,3,4,5] |
```

## bitmapXor

Two bitmap xor calculation, the result is a new bitmap.

```
bitmapXor(bitmap,bitmap)
```

### Arguments

- `bitmap` – Bitmap object.

### Example

```
SELECT bitmapToArray(bitmapXor(bitmapBuild([1,2,3]),bitmapBuild([3,4,5]))) AS res;
```

```
res  
[1,2,4,5] |
```

## bitmapAndnot

Two bitmap andnot calculation, the result is a new bitmap.

```
bitmapAndnot(bitmap,bitmap)
```

### Arguments

- `bitmap` – Bitmap object.

### Example

```
SELECT bitmapToArray(bitmapAndnot(bitmapBuild([1,2,3]),bitmapBuild([3,4,5]))) AS res;
```

```
res  
[1,2] |
```

## bitmapAndCardinality

Two bitmap and calculation, return cardinality of type UInt64.

```
bitmapAndCardinality(bitmap,bitmap)
```

### Arguments

- `bitmap` – Bitmap object.

### Example

```
SELECT bitmapAndCardinality(bitmapBuild([1,2,3]),bitmapBuild([3,4,5])) AS res;
```

```
res  
1 |
```

## bitmapOrCardinality

Two bitmap or calculation, return cardinality of type UInt64.

```
bitmapOrCardinality(bitmap,bitmap)
```

### Arguments

- `bitmap` – Bitmap object.

### Example

```
SELECT bitmapOrCardinality(bitmapBuild([1,2,3]),bitmapBuild([3,4,5])) AS res;
```

```
res  
5 |
```

## bitmapXorCardinality

Two bitmap xor calculation, return cardinality of type UInt64.

```
bitmapXorCardinality(bitmap,bitmap)
```

### Arguments

- `bitmap` – Bitmap object.

### Example

```
SELECT bitmapXorCardinality(bitmapBuild([1,2,3]),bitmapBuild([3,4,5])) AS res;
```

```
res  
4 |
```

## bitmapAndnotCardinality

Two bitmap andnot calculation, return cardinality of type UInt64.

```
bitmapAndnotCardinality(bitmap,bitmap)
```

### Arguments

- bitmap – Bitmap object.

### Example

```
SELECT bitmapAndnotCardinality(bitmapBuild([1,2,3]),bitmapBuild([3,4,5])) AS res;
```

```
res  
2 |
```

## Hash Functions

Hash functions can be used for the deterministic pseudo-random shuffling of elements.

Simhash is a hash function, which returns close hash values for close (similar) arguments.

## halfMD5

**Interprets** all the input parameters as strings and calculates the **MD5** hash value for each of them. Then combines hashes, takes the first 8 bytes of the hash of the resulting string, and interprets them as **UInt64** in big-endian byte order.

```
halfMD5(par1, ...)
```

The function is relatively slow (5 million short strings per second per processor core).

Consider using the **sipHash64** function instead.

### Arguments

The function takes a variable number of input parameters. Arguments can be any of the **supported data types**.

### Returned Value

A **UInt64** data type hash value.

### Example

```
SELECT halfMD5(array('e','x','a'), 'mple', 10, toDateTime('2019-06-15 23:00:00')) AS halfMD5hash,  
toTypeName(halfMD5hash) AS type;
```

halfMD5hash	type
186182704141653334	UInt64

## MD4

Calculates the MD4 from a string and returns the resulting set of bytes as FixedString(16).

## MD5

Calculates the MD5 from a string and returns the resulting set of bytes as FixedString(16). If you do not need MD5 in particular, but you need a decent cryptographic 128-bit hash, use the 'sipHash128' function instead. If you want to get the same result as output by the md5sum utility, use lower(hex(MD5(s))).

## sipHash64

Produces a 64-bit [SipHash](#) hash value.

```
sipHash64(par1,...)
```

This is a cryptographic hash function. It works at least three times faster than the [MD5](#) function.

Function [interprets](#) all the input parameters as strings and calculates the hash value for each of them. Then combines hashes by the following algorithm:

1. After hashing all the input parameters, the function gets the array of hashes.
2. Function takes the first and the second elements and calculates a hash for the array of them.
3. Then the function takes the hash value, calculated at the previous step, and the third element of the initial hash array, and calculates a hash for the array of them.
4. The previous step is repeated for all the remaining elements of the initial hash array.

### Arguments

The function takes a variable number of input parameters. Arguments can be any of the [supported data types](#).

### Returned Value

A [UInt64](#) data type hash value.

### Example

```
SELECT sipHash64(array('e','x','a'), 'mple', 10, toDateTime('2019-06-15 23:00:00')) AS SipHash, toTypeName(SipHash)  
AS type;
```

SipHash	type
13726873534472839665	UInt64

## sipHash128

Calculates SipHash from a string.

Accepts a String-type argument. Returns FixedString(16).

Differs from sipHash64 in that the final xor-folding state is only done up to 128 bits.

## cityHash64

Produces a 64-bit [CityHash](#) hash value.

```
cityHash64(par1,...)
```

This is a fast non-cryptographic hash function. It uses the CityHash algorithm for string parameters and implementation-specific fast non-cryptographic hash function for parameters with other data types. The function uses the CityHash combinator to get the final results.

### Arguments

The function takes a variable number of input parameters. Arguments can be any of the [supported data types](#).

### Returned Value

A [UInt64](#) data type hash value.

### Examples

Call example:

```
SELECT cityHash64(array('e','x','a'), 'mple', 10, toDateTime('2019-06-15 23:00:00')) AS CityHash,  
toTypeName(CityHash) AS type;
```

CityHash	type
12072650598913549138	UInt64

The following example shows how to compute the checksum of the entire table with accuracy up to the row order:

```
SELECT groupBitXor(cityHash64(*)) FROM table
```

## intHash32

Calculates a 32-bit hash code from any type of integer.

This is a relatively fast non-cryptographic hash function of average quality for numbers.

## intHash64

Calculates a 64-bit hash code from any type of integer.

It works faster than intHash32. Average quality.

## SHA1, SHA224, SHA256, SHA512

Calculates SHA-1, SHA-224, SHA-256, SHA-512 hash from a string and returns the resulting set of bytes as [FixedString](#).

### Syntax

```
SHA1('s')
...
SHA512('s')
```

The function works fairly slowly (SHA-1 processes about 5 million short strings per second per processor core, while SHA-224 and SHA-256 process about 2.2 million).

We recommend using this function only in cases when you need a specific hash function and you can't select it.

Even in these cases, we recommend applying the function offline and pre-calculating values when inserting them into the table, instead of applying it in `SELECT` queries.

## Arguments

- `s` — Input string for SHA hash calculation. [String](#).

## Returned value

- SHA hash as a hex-unencoded `FixedString`. SHA-1 returns as `FixedString(20)`, SHA-224 as `FixedString(28)`, SHA-256 — `FixedString(32)`, SHA-512 — `FixedString(64)`.

Type: [FixedString](#).

## Example

Use the `hex` function to represent the result as a hex-encoded string.

Query:

```
SELECT hex(SHA1('abc'));
```

Result:

```
hex(SHA1('abc'))—————
A9993E364706816ABA3E25717850C26C9CD0D89D |
```

## URLHash(url[, N])

A fast, decent-quality non-cryptographic hash function for a string obtained from a URL using some type of normalization.

`URLHash(s)` – Calculates a hash from a string without one of the trailing symbols `/?` or `#` at the end, if present.

`URLHash(s, N)` – Calculates a hash from a string up to the N level in the URL hierarchy, without one of the trailing symbols `/?` or `#` at the end, if present.

Levels are the same as in `URLHierarchy`. This function is specific to Yandex.Metrica.

## farmFingerprint64

## farmHash64

Produces a 64-bit [FarmHash](#) or Fingerprint value. `farmFingerprint64` is preferred for a stable and portable value.

```
farmFingerprint64(par1, ...)
farmHash64(par1, ...)
```

These functions use the `Fingerprint64` and `Hash64` methods respectively from all [available methods](#).

## Arguments

The function takes a variable number of input parameters. Arguments can be any of the [supported data types](#).

## Returned Value

A `UInt64` data type hash value.

## Example

```
SELECT farmHash64(array('e','x','a'), 'mple', 10, toDateTime('2019-06-15 23:00:00')) AS FarmHash,  
toTypeName(FarmHash) AS type;
```

17790458267262532859	UInt64
FarmHash	type

# javaHash

Calculates [JavaHash](#) from a string. This hash function is neither fast nor having a good quality. The only reason to use it is when this algorithm is already used in another system and you have to calculate exactly the same result.

## Syntax

```
SELECT javaHash("")
```

## Returned value

A `Int32` data type hash value.

## Example

Query:

```
SELECT javaHash('Hello, world!');
```

Result:

javaHash('Hello, world!')	-1880044555
---------------------------	-------------

# javaHashUTF16LE

Calculates [JavaHash](#) from a string, assuming it contains bytes representing a string in UTF-16LE encoding.

## Syntax

```
javaHashUTF16LE(stringUtf16le)
```

## Arguments

- `stringUtf16le` — a string in UTF-16LE encoding.

## Returned value

A `Int32` data type hash value.

## Example

Correct query with UTF-16LE encoded string.

Query:

```
SELECT javaHashUTF16LE(convertCharset('test', 'utf-8', 'utf-16le'));
```

Result:

```
javaHashUTF16LE(convertCharset('test', 'utf-8', 'utf-16le'))—  
3556498 |
```

## hiveHash

Calculates `HiveHash` from a string.

```
SELECT hiveHash("")
```

This is just `JavaHash` with zeroed out sign bit. This function is used in [Apache Hive](#) for versions before 3.0. This hash function is neither fast nor having a good quality. The only reason to use it is when this algorithm is already used in another system and you have to calculate exactly the same result.

## Returned value

A `Int32` data type hash value.

Type: `hiveHash`.

## Example

Query:

```
SELECT hiveHash('Hello, world!');
```

Result:

```
hiveHash('Hello, world!')—  
267439093 |
```

## metroHash64

Produces a 64-bit [MetroHash](#) hash value.

```
metroHash64(par1, ...)
```

## Arguments

The function takes a variable number of input parameters. Arguments can be any of the [supported data types](#).

## Returned Value

A [UInt64](#) data type hash value.

## Example

```
SELECT metroHash64(array('e','x','a'), 'mple', 10, toDateTime('2019-06-15 23:00:00')) AS MetroHash,  
toTypeName(MetroHash) AS type;
```

MetroHash	type
14235658766382344533	UInt64

## jumpConsistentHash

Calculates JumpConsistentHash form a UInt64.

Accepts two arguments: a UInt64-type key and the number of buckets. Returns Int32.

For more information, see the link: [JumpConsistentHash](#)

## murmurHash2\_32, murmurHash2\_64

Produces a [MurmurHash2](#) hash value.

```
murmurHash2_32(par1, ...)  
murmurHash2_64(par1, ...)
```

## Arguments

Both functions take a variable number of input parameters. Arguments can be any of the [supported data types](#).

## Returned Value

- The `murmurHash2_32` function returns hash value having the [UInt32](#) data type.
- The `murmurHash2_64` function returns hash value having the [UInt64](#) data type.

## Example

```
SELECT murmurHash2_64(array('e','x','a'), 'mple', 10, toDateTime('2019-06-15 23:00:00')) AS MurmurHash2,  
toTypeName(MurmurHash2) AS type;
```

MurmurHash2	type
11832096901709403633	UInt64

## gccMurmurHash

Calculates a 64-bit [MurmurHash2](#) hash value using the same hash seed as [gcc](#). It is portable between CLang and GCC builds.

## Syntax

```
gccMurmurHash(par1, ...)
```

## Arguments

- par1, ... — A variable number of parameters that can be any of the [supported data types](#).

## Returned value

- Calculated hash value.

Type: [UInt64](#).

## Example

Query:

```
SELECT
    gccMurmurHash(1, 2, 3) AS res1,
    gccMurmurHash(['a', [1, 2, 3], 4, (4, ['foo', 'bar']), 1, (1, 2))) AS res2
```

Result:

res1	res2
12384823029245979431	1188926775431157506

## murmurHash3\_32, murmurHash3\_64

Produces a [MurmurHash3](#) hash value.

```
murmurHash3_32(par1, ...)
murmurHash3_64(par1, ...)
```

## Arguments

Both functions take a variable number of input parameters. Arguments can be any of the [supported data types](#).

## Returned Value

- The `murmurHash3_32` function returns a [UInt32](#) data type hash value.
- The `murmurHash3_64` function returns a [UInt64](#) data type hash value.

## Example

```
SELECT murmurHash3_32(array('e','x','a'), 'mple', 10, toDateTime('2019-06-15 23:00:00')) AS MurmurHash3,
toTypeName(MurmurHash3) AS type;
```

MurmurHash3	type
2152717	UInt32

## murmurHash3\_128

Produces a 128-bit [MurmurHash3](#) hash value.

```
murmurHash3_128( expr )
```

## Arguments

- `expr` — [Expressions](#) returning a [String](#)-type value.

## Returned Value

A [FixedString\(16\)](#) data type hash value.

## Example

```
SELECT hex(murmurHash3_128('example_string')) AS MurmurHash3, toTypeName(MurmurHash3) AS type;
```

MurmurHash3	368A1A311CB7342253354B548E7E7E71	String
-------------	----------------------------------	--------

## xxHash32, xxHash64

Calculates `xxHash` from a string. It is proposed in two flavors, 32 and 64 bits.

```
SELECT xxHash32("")
```

OR

```
SELECT xxHash64("")
```

## Returned value

A `Uint32` or `Uint64` data type hash value.

Type: `xxHash`.

## Example

Query:

```
SELECT xxHash32('Hello, world!');
```

Result:

xxHash32('Hello, world!')	834093149
---------------------------	-----------

## See Also

- [xxHash](#).

## ngramSimHash

Splits a ASCII string into n-grams of `ngramsize` symbols and returns the n-gram simhash. Is case sensitive.

Can be used for detection of semi-duplicate strings with [bitHammingDistance](#). The smaller is the [Hamming Distance](#) of the calculated simhashes of two strings, the more likely these strings are the same.

## Syntax

```
ngramSimHash(string[, ngramsize])
```

## Arguments

- `string` — String. [String](#).
- `ngramsize` — The size of an n-gram. Optional. Possible values: any number from [1](#) to [25](#). Default value: [3](#). [UInt8](#).

## Returned value

- Hash value.

Type: [UInt64](#).

## Example

Query:

```
SELECT ngramSimHash('ClickHouse') AS Hash;
```

Result:

```
+-----+  
| Hash |  
+-----+  
| 1627567969 |  
+-----+
```

## ngramSimHashCaseInsensitive

Splits a ASCII string into n-grams of `ngramsize` symbols and returns the n-gram `simhash`. Is case insensitive.

Can be used for detection of semi-duplicate strings with [bitHammingDistance](#). The smaller is the [Hamming Distance](#) of the calculated simhashes of two strings, the more likely these strings are the same.

## Syntax

```
ngramSimHashCaseInsensitive(string[, ngramsize])
```

## Arguments

- `string` — String. [String](#).
- `ngramsize` — The size of an n-gram. Optional. Possible values: any number from [1](#) to [25](#). Default value: [3](#). [UInt8](#).

## Returned value

- Hash value.

Type: [UInt64](#).

## Example

Query:

```
SELECT ngramSimHashCaseInsensitive('ClickHouse') AS Hash;
```

Result:

```
Hash  
562180645
```

## ngramSimHashUTF8

Splits a UTF-8 string into n-grams of `ngramsize` symbols and returns the n-gram simhash. Is case sensitive.

Can be used for detection of semi-duplicate strings with [bitHammingDistance](#). The smaller is the [Hamming Distance](#) of the calculated simhashes of two strings, the more likely these strings are the same.

### Syntax

```
ngramSimHashUTF8(string[, ngramsize])
```

### Arguments

- `string` — String. [String](#).
- `ngramsize` — The size of an n-gram. Optional. Possible values: any number from [1](#) to [25](#). Default value: [3](#). [UInt8](#).

### Returned value

- Hash value.

Type: [UInt64](#).

### Example

Query:

```
SELECT ngramSimHashUTF8('ClickHouse') AS Hash;
```

Result:

```
Hash  
1628157797
```

## ngramSimHashCaseInsensitiveUTF8

Splits a UTF-8 string into n-grams of `ngramsize` symbols and returns the n-gram simhash. Is case insensitive.

Can be used for detection of semi-duplicate strings with [bitHammingDistance](#). The smaller is the [Hamming Distance](#) of the calculated simhashes of two strings, the more likely these strings are the same.

### Syntax

```
ngramSimHashCaseInsensitiveUTF8(string[, ngramsize])
```

### Arguments

- `string` — String. [String](#).
- `ngramsize` — The size of an n-gram. Optional. Possible values: any number from `1` to `25`. Default value: `3`. [UInt8](#).

## Returned value

- Hash value.

Type: [UInt64](#).

## Example

Query:

```
SELECT ngramSimHashCaseInsensitiveUTF8('ClickHouse') AS Hash;
```

Result:

Hash
1636742693

# wordShingleSimHash

Splits a ASCII string into parts (shingles) of `shinglesize` words and returns the word shingle `simhash`. Is case sensitive.

Can be used for detection of semi-duplicate strings with [bitHammingDistance](#). The smaller is the [Hamming Distance](#) of the calculated `simhashes` of two strings, the more likely these strings are the same.

## Syntax

```
wordShingleSimHash(string[, shinglesize])
```

## Arguments

- `string` — String. [String](#).
- `shinglesize` — The size of a word shingle. Optional. Possible values: any number from `1` to `25`. Default value: `3`. [UInt8](#).

## Returned value

- Hash value.

Type: [UInt64](#).

## Example

Query:

```
SELECT wordShingleSimHash('ClickHouse® is a column-oriented database management system (DBMS) for online analytical processing of queries (OLAP).') AS Hash;
```

Result:

Hash
2328277067

## wordShingleSimHashCaseInsensitive

Splits a ASCII string into parts (shingles) of shinglesize words and returns the word shingle simhash. Is case insensitive.

Can be used for detection of semi-duplicate strings with [bitHammingDistance](#). The smaller is the [Hamming Distance](#) of the calculated simhashes of two strings, the more likely these strings are the same.

### Syntax

```
wordShingleSimHashCaseInsensitive(string[, shinglesize])
```

### Arguments

- `string` — String. [String](#).
- `shinglesize` — The size of a word shingle. Optional. Possible values: any number from `1` to `25`. Default value: `3`. [UInt8](#).

### Returned value

- Hash value.

Type: [UInt64](#).

### Example

Query:

```
SELECT wordShingleSimHashCaseInsensitive('ClickHouse® is a column-oriented database management system  
(DBMS) for online analytical processing of queries (OLAP).') AS Hash;
```

Result:

Hash
2194812424

## wordShingleSimHashUTF8

Splits a UTF-8 string into parts (shingles) of shinglesize words and returns the word shingle simhash. Is case sensitive.

Can be used for detection of semi-duplicate strings with [bitHammingDistance](#). The smaller is the [Hamming Distance](#) of the calculated simhashes of two strings, the more likely these strings are the same.

### Syntax

```
wordShingleSimHashUTF8(string[, shinglesize])
```

### Arguments

- `string` — String. [String](#).

- `shinglesize` — The size of a word shingle. Optional. Possible values: any number from 1 to 25. Default value: 3. [UInt8](#).

## Returned value

- Hash value.

Type: [UInt64](#).

## Example

Query:

```
SELECT wordShingleSimHashUTF8('ClickHouse® is a column-oriented database management system (DBMS) for online analytical processing of queries (OLAP).') AS Hash;
```

Result:

Hash
2328277067

## wordShingleSimHashCaseInsensitiveUTF8

Splits a UTF-8 string into parts (shingles) of shinglesize words and returns the word shingle simhash. Is case insensitive.

Can be used for detection of semi-duplicate strings with [bitHammingDistance](#). The smaller is the [Hamming Distance](#) of the calculated simhashes of two strings, the more likely these strings are the same.

## Syntax

```
wordShingleSimHashCaseInsensitiveUTF8(string[, shinglesize])
```

## Arguments

- `string` — String. [String](#).
- `shinglesize` — The size of a word shingle. Optional. Possible values: any number from 1 to 25. Default value: 3. [UInt8](#).

## Returned value

- Hash value.

Type: [UInt64](#).

## Example

Query:

```
SELECT wordShingleSimHashCaseInsensitiveUTF8('ClickHouse® is a column-oriented database management system (DBMS) for online analytical processing of queries (OLAP).') AS Hash;
```

Result:

```
Hash  
2194812424
```

## ngramMinHash

Splits a ASCII string into n-grams of `ngramsize` symbols and calculates hash values for each n-gram. Uses `hashnum` minimum hashes to calculate the minimum hash and `hashnum` maximum hashes to calculate the maximum hash. Returns a tuple with these hashes. Is case sensitive.

Can be used for detection of semi-duplicate strings with [tupleHammingDistance](#). For two strings: if one of the returned hashes is the same for both strings, we think that those strings are the same.

### Syntax

```
ngramMinHash(string[, ngramsize, hashnum])
```

### Arguments

- `string` — String. [String](#).
- `ngramsize` — The size of an n-gram. Optional. Possible values: any number from `1` to `25`. Default value: `3`. [UInt8](#).
- `hashnum` — The number of minimum and maximum hashes used to calculate the result. Optional. Possible values: any number from `1` to `25`. Default value: `6`. [UInt8](#).

### Returned value

- Tuple with two hashes — the minimum and the maximum.

Type: [Tuple\(UInt64, UInt64\)](#).

### Example

Query:

```
SELECT ngramMinHash('ClickHouse') AS Tuple;
```

Result:

```
Tuple—  
(18333312859352735453,905424844481805918)
```

## ngramMinHashCaseInsensitive

Splits a ASCII string into n-grams of `ngramsize` symbols and calculates hash values for each n-gram. Uses `hashnum` minimum hashes to calculate the minimum hash and `hashnum` maximum hashes to calculate the maximum hash. Returns a tuple with these hashes. Is case insensitive.

Can be used for detection of semi-duplicate strings with [tupleHammingDistance](#). For two strings: if one of the returned hashes is the same for both strings, we think that those strings are the same.

### Syntax

```
ngramMinHashCaseInsensitive(string[, ngramsize, hashnum])
```

## Arguments

- `string` — String. [String](#).
- `ngramsize` — The size of an n-gram. Optional. Possible values: any number from 1 to 25. Default value: 3. [UInt8](#).
- `hashnum` — The number of minimum and maximum hashes used to calculate the result. Optional. Possible values: any number from 1 to 25. Default value: 6. [UInt8](#).

## Returned value

- Tuple with two hashes — the minimum and the maximum.

Type: [Tuple\(UInt64, UInt64\)](#).

## Example

Query:

```
SELECT ngramMinHashCaseInsensitive('ClickHouse') AS Tuple;
```

Result:

```
Tuple  
 (2106263556442004574,13203602793651726206)
```

## ngramMinHashUTF8

Splits a UTF-8 string into n-grams of `ngramsize` symbols and calculates hash values for each n-gram. Uses `hashnum` minimum hashes to calculate the minimum hash and `hashnum` maximum hashes to calculate the maximum hash. Returns a tuple with these hashes. Is case sensitive.

Can be used for detection of semi-duplicate strings with [tupleHammingDistance](#). For two strings: if one of the returned hashes is the same for both strings, we think that those strings are the same.

## Syntax

```
ngramMinHashUTF8(string[, ngramsize, hashnum])
```

## Arguments

- `string` — String. [String](#).
- `ngramsize` — The size of an n-gram. Optional. Possible values: any number from 1 to 25. Default value: 3. [UInt8](#).
- `hashnum` — The number of minimum and maximum hashes used to calculate the result. Optional. Possible values: any number from 1 to 25. Default value: 6. [UInt8](#).

## Returned value

- Tuple with two hashes — the minimum and the maximum.

Type: [Tuple\(UInt64, UInt64\)](#).

## Example

Query:

```
SELECT ngramMinHashUTF8('ClickHouse') AS Tuple;
```

Result:

```
Tuple  
(18333312859352735453,6742163577938632877)
```

## ngramMinHashCaseInsensitiveUTF8

Splits a UTF-8 string into n-grams of ngramsize symbols and calculates hash values for each n-gram. Uses hashnum minimum hashes to calculate the minimum hash and hashnum maximum hashes to calculate the maximum hash. Returns a tuple with these hashes. Is case insensitive.

Can be used for detection of semi-duplicate strings with [tupleHammingDistance](#). For two strings: if one of the returned hashes is the same for both strings, we think that those strings are the same.

## Syntax

```
ngramMinHashCaseInsensitiveUTF8(string [, ngramsize, hashnum])
```

## Arguments

- `string` — String. [String](#).
- `ngramsize` — The size of an n-gram. Optional. Possible values: any number from `1` to `25`. Default value: `3`. [UInt8](#).
- `hashnum` — The number of minimum and maximum hashes used to calculate the result. Optional. Possible values: any number from `1` to `25`. Default value: `6`. [UInt8](#).

## Returned value

- Tuple with two hashes — the minimum and the maximum.

Type: [Tuple\(UInt64, UInt64\)](#).

## Example

Query:

```
SELECT ngramMinHashCaseInsensitiveUTF8('ClickHouse') AS Tuple;
```

Result:

```
Tuple  
(12493625717655877135,13203602793651726206)
```

## ngramMinHashArg

Splits a ASCII string into n-grams of ngramsize symbols and returns the n-grams with minimum and maximum hashes, calculated by the [ngramMinHash](#) function with the same input. Is case sensitive.

## Syntax

```
ngramMinHashArg(string[, ngramsize, hashnum])
```

## Arguments

- `string` — String. [String](#).
- `ngramsize` — The size of an n-gram. Optional. Possible values: any number from `1` to `25`. Default value: `3`. [UInt8](#).
- `hashnum` — The number of minimum and maximum hashes used to calculate the result. Optional. Possible values: any number from `1` to `25`. Default value: `6`. [UInt8](#).

## Returned value

- Tuple with two tuples with `hashnum` n-grams each.

Type: [Tuple\(Tuple\(String\), Tuple\(String\)\)](#).

## Example

Query:

```
SELECT ngramMinHashArg('ClickHouse') AS Tuple;
```

Result:

```
Tuple
(( 'ous' , 'ick' , 'lic' , 'Hou' , 'kHo' , 'use' ), ( 'Hou' , 'lic' , 'ick' , 'ous' , 'ckH' , 'Cli' ) ) |
```

## ngramMinHashArgCaseInsensitive

Splits a ASCII string into n-grams of `ngramsize` symbols and returns the n-grams with minimum and maximum hashes, calculated by the [ngramMinHashCaseInsensitive](#) function with the same input. Is case insensitive.

## Syntax

```
ngramMinHashArgCaseInsensitive(string[, ngramsize, hashnum])
```

## Arguments

- `string` — String. [String](#).
- `ngramsize` — The size of an n-gram. Optional. Possible values: any number from `1` to `25`. Default value: `3`. [UInt8](#).
- `hashnum` — The number of minimum and maximum hashes used to calculate the result. Optional. Possible values: any number from `1` to `25`. Default value: `6`. [UInt8](#).

## Returned value

- Tuple with two tuples with `hashnum` n-grams each.

Type: [Tuple\(Tuple\(String\), Tuple\(String\)\)](#).

## Example

Query:

```
SELECT ngramMinHashArgCaseInsensitive('ClickHouse') AS Tuple;
```

Result:

```
Tuple
((‘ous’, ‘ick’, ‘lic’, ‘kHo’, ‘use’, ‘Cli’), (‘kHo’, ‘lic’, ‘ick’, ‘ous’, ‘ckH’, ‘Hou’)) |
```

## ngramMinHashArgUTF8

Splits a UTF-8 string into n-grams of ngramsize symbols and returns the n-grams with minimum and maximum hashes, calculated by the [ngramMinHashUTF8](#) function with the same input. Is case sensitive.

### Syntax

```
ngramMinHashArgUTF8(string[, ngramsize, hashnum])
```

### Arguments

- `string` — String. [String](#).
- `ngramsize` — The size of an n-gram. Optional. Possible values: any number from `1` to `25`. Default value: `3`. [UInt8](#).
- `hashnum` — The number of minimum and maximum hashes used to calculate the result. Optional. Possible values: any number from `1` to `25`. Default value: `6`. [UInt8](#).

### Returned value

- Tuple with two tuples with `hashnum` n-grams each.

Type: [Tuple\(Tuple\(String\), Tuple\(String\)\)](#).

## Example

Query:

```
SELECT ngramMinHashArgUTF8('ClickHouse') AS Tuple;
```

Result:

```
Tuple
((‘ous’, ‘ick’, ‘lic’, ‘Hou’, ‘kHo’, ‘use’), (‘kHo’, ‘Hou’, ‘lic’, ‘ick’, ‘ous’, ‘ckH’)) |
```

## ngramMinHashArgCaseInsensitiveUTF8

Splits a UTF-8 string into n-grams of ngramsize symbols and returns the n-grams with minimum and maximum hashes, calculated by the [ngramMinHashCaseInsensitiveUTF8](#) function with the same input. Is case insensitive.

### Syntax

```
ngramMinHashArgCaseInsensitiveUTF8(string[], ngramsize, hashnum)
```

## Arguments

- `string` — String. [String](#).
- `ngramsize` — The size of an n-gram. Optional. Possible values: any number from `1` to `25`. Default value: `3`. [UInt8](#).
- `hashnum` — The number of minimum and maximum hashes used to calculate the result. Optional. Possible values: any number from `1` to `25`. Default value: `6`. [UInt8](#).

## Returned value

- Tuple with two tuples with `hashnum` n-grams each.

Type: [Tuple\(Tuple\(String\), Tuple\(String\)\)](#).

## Example

Query:

```
SELECT ngramMinHashArgCaseInsensitiveUTF8('ClickHouse') AS Tuple;
```

Result:

```
Tuple
((`ckH`, `ous`, `ick`, `lic`, `kHo`, `use`), (`kHo`, `lic`, `ick`, `ous`, `ckH`, `Hou`)) |
```

## wordShingleMinHash

Splits a ASCII string into parts (shingles) of `shinglesize` words and calculates hash values for each word shingle. Uses `hashnum` minimum hashes to calculate the minimum hash and `hashnum` maximum hashes to calculate the maximum hash. Returns a tuple with these hashes. Is case sensitive.

Can be used for detection of semi-duplicate strings with [tupleHammingDistance](#). For two strings: if one of the returned hashes is the same for both strings, we think that those strings are the same.

## Syntax

```
wordShingleMinHash(string[], shinglesize, hashnum)
```

## Arguments

- `string` — String. [String](#).
- `shinglesize` — The size of a word shingle. Optional. Possible values: any number from `1` to `25`. Default value: `3`. [UInt8](#).
- `hashnum` — The number of minimum and maximum hashes used to calculate the result. Optional. Possible values: any number from `1` to `25`. Default value: `6`. [UInt8](#).

## Returned value

- Tuple with two hashes — the minimum and the maximum.

Type: [Tuple\(UInt64, UInt64\)](#).

## Example

Query:

```
SELECT wordShingleMinHash('ClickHouse® is a column-oriented database management system (DBMS) for online analytical processing of queries (OLAP).') AS Tuple;
```

Result:

```
Tuple  
(16452112859864147620,5844417301642981317)
```

## wordShingleMinHashCaseInsensitive

Splits a ASCII string into parts (shingles) of shingleSize words and calculates hash values for each word shingle. Uses `hashnum_min` minimum hashes to calculate the minimum hash and `hashnum_max` maximum hashes to calculate the maximum hash. Returns a tuple with these hashes. Is case insensitive.

Can be used for detection of semi-duplicate strings with [tupleHammingDistance](#). For two strings: if one of the returned hashes is the same for both strings, we think that those strings are the same.

## Syntax

```
wordShingleMinHashCaseInsensitive(string[, shingleSize, hashnum])
```

## Arguments

- `string` — String. [String](#).
- `shingleSize` — The size of a word shingle. Optional. Possible values: any number from `1` to `25`. Default value: `3`. [UInt8](#).
- `hashnum` — The number of minimum and maximum hashes used to calculate the result. Optional. Possible values: any number from `1` to `25`. Default value: `6`. [UInt8](#).

## Returned value

- Tuple with two hashes — the minimum and the maximum.

Type: [Tuple\(UInt64, UInt64\)](#).

## Example

Query:

```
SELECT wordShingleMinHashCaseInsensitive('ClickHouse® is a column-oriented database management system (DBMS) for online analytical processing of queries (OLAP).') AS Tuple;
```

Result:

```
Tuple  
(3065874883688416519,1634050779997673240)
```

## wordShingleMinHashUTF8

Splits a UTF-8 string into parts (shingles) of shingle size words and calculates hash values for each word shingle. Uses hashnum minimum hashes to calculate the minimum hash and hashnum maximum hashes to calculate the maximum hash. Returns a tuple with these hashes. Is case sensitive.

Can be used for detection of semi-duplicate strings with [tupleHammingDistance](#). For two strings: if one of the returned hashes is the same for both strings, we think that those strings are the same.

## Syntax

```
wordShingleMinHashUTF8(string[, shingleSize, hashNum])
```

## Arguments

- `string` — String. [String](#).
- `shingleSize` — The size of a word shingle. Optional. Possible values: any number from 1 to 25. Default value: 3. [UInt8](#).
- `hashNum` — The number of minimum and maximum hashes used to calculate the result. Optional. Possible values: any number from 1 to 25. Default value: 6. [UInt8](#).

## Returned value

- Tuple with two hashes — the minimum and the maximum.

Type: [Tuple\(UInt64, UInt64\)](#).

## Example

Query:

```
SELECT wordShingleMinHashUTF8('ClickHouse® is a column-oriented database management system (DBMS) for online analytical processing of queries (OLAP).') AS Tuple;
```

Result:

```
Tuple  
 (16452112859864147620,5844417301642981317)
```

## wordShingleMinHashCaseInsensitiveUTF8

Splits a UTF-8 string into parts (shingles) of shingle size words and calculates hash values for each word shingle. Uses hashnum minimum hashes to calculate the minimum hash and hashnum maximum hashes to calculate the maximum hash. Returns a tuple with these hashes. Is case insensitive.

Can be used for detection of semi-duplicate strings with [tupleHammingDistance](#). For two strings: if one of the returned hashes is the same for both strings, we think that those strings are the same.

## Syntax

```
wordShingleMinHashCaseInsensitiveUTF8(string[, shingleSize, hashNum])
```

## Arguments

- `string` — String. [String](#).

- `shinglesize` — The size of a word shingle. Optional. Possible values: any number from 1 to 25. Default value: 3. [UInt8](#).
- `hashnum` — The number of minimum and maximum hashes used to calculate the result. Optional. Possible values: any number from 1 to 25. Default value: 6. [UInt8](#).

## Returned value

- Tuple with two hashes — the minimum and the maximum.

Type: [Tuple\(UInt64, UInt64\)](#).

## Example

Query:

```
SELECT wordShingleMinHashCaseInsensitiveUTF8('ClickHouse® is a column-oriented database management system  
(DBMS) for online analytical processing of queries (OLAP).') AS Tuple;
```

Result:

```
Tuple  
(3065874883688416519,1634050779997673240)
```

## wordShingleMinHashArg

Splits a ASCII string into parts (shingles) of `shinglesize` words each and returns the shingles with minimum and maximum word hashes, calculated by the [wordShingleMinHash](#) function with the same input. Is case sensitive.

## Syntax

```
wordShingleMinHashArg(string[, shinglesize, hashnum])
```

## Arguments

- `string` — String. [String](#).
- `shinglesize` — The size of a word shingle. Optional. Possible values: any number from 1 to 25. Default value: 3. [UInt8](#).
- `hashnum` — The number of minimum and maximum hashes used to calculate the result. Optional. Possible values: any number from 1 to 25. Default value: 6. [UInt8](#).

## Returned value

- Tuple with two tuples with `hashnum` word shingles each.

Type: [Tuple\(Tuple\(String\), Tuple\(String\)\)](#).

## Example

Query:

```
SELECT wordShingleMinHashArg('ClickHouse® is a column-oriented database management system (DBMS) for online  
analytical processing of queries (OLAP).', 1, 3) AS Tuple;
```

Result:

```
Tuple
((OLAP', 'database', 'analytical'), ('online', 'oriented', 'processing')) |
```

## wordShingleMinHashArgCaseInsensitive

Splits a ASCII string into parts (shingles) of shinglesize words each and returns the shingles with minimum and maximum word hashes, calculated by the [wordShingleMinHashCaseInsensitive](#) function with the same input. Is case insensitive.

### Syntax

```
wordShingleMinHashArgCaseInsensitive(string[, shinglesize, hashnum])
```

### Arguments

- `string` — String. [String](#).
- `shinglesize` — The size of a word shingle. Optional. Possible values: any number from `1` to `25`. Default value: `3`. [UInt8](#).
- `hashnum` — The number of minimum and maximum hashes used to calculate the result. Optional. Possible values: any number from `1` to `25`. Default value: `6`. [UInt8](#).

### Returned value

- Tuple with two tuples with `hashnum` word shingles each.

Type: [Tuple\(Tuple\(String\), Tuple\(String\)\)](#).

### Example

Query:

```
SELECT wordShingleMinHashArgCaseInsensitive('ClickHouse® is a column-oriented database management system (DBMS) for online analytical processing of queries (OLAP).', 1, 3) AS Tuple;
```

Result:

```
Tuple
(("queries", "database", "analytical"), ("oriented", "processing", "DBMS")) |
```

## wordShingleMinHashArgUTF8

Splits a UTF-8 string into parts (shingles) of shinglesize words each and returns the shingles with minimum and maximum word hashes, calculated by the [wordShingleMinHashUTF8](#) function with the same input. Is case sensitive.

### Syntax

```
wordShingleMinHashArgUTF8(string[, shinglesize, hashnum])
```

### Arguments

- `string` — String. [String](#).
- `shinglesize` — The size of a word shingle. Optional. Possible values: any number from `1` to `25`. Default value: `3`. [UInt8](#).
- `hashnum` — The number of minimum and maximum hashes used to calculate the result. Optional. Possible values: any number from `1` to `25`. Default value: `6`. [UInt8](#).

### Returned value

- Tuple with two tuples with `hashnum` word shingles each.

Type: [Tuple\(Tuple\(String\), Tuple\(String\)\)](#).

### Example

Query:

```
SELECT wordShingleMinHashArgUTF8('ClickHouse® is a column-oriented database management system (DBMS) for online analytical processing of queries (OLAP).', 1, 3) AS Tuple;
```

Result:

```
Tuple
((('OLAP','database','analytical'),('online','oriented','processing')) |
```

## wordShingleMinHashArgCaseInsensitiveUTF8

Splits a UTF-8 string into parts (shingles) of `shinglesize` words each and returns the shingles with minimum and maximum word hashes, calculated by the [wordShingleMinHashCaseInsensitiveUTF8](#) function with the same input. Is case insensitive.

### Syntax

```
wordShingleMinHashArgCaseInsensitiveUTF8(string[, shinglesize, hashnum])
```

### Arguments

- `string` — String. [String](#).
- `shinglesize` — The size of a word shingle. Optional. Possible values: any number from `1` to `25`. Default value: `3`. [UInt8](#).
- `hashnum` — The number of minimum and maximum hashes used to calculate the result. Optional. Possible values: any number from `1` to `25`. Default value: `6`. [UInt8](#).

### Returned value

- Tuple with two tuples with `hashnum` word shingles each.

Type: [Tuple\(Tuple\(String\), Tuple\(String\)\)](#).

### Example

Query:

```
SELECT wordShingleMinHashArgCaseInsensitiveUTF8('ClickHouse® is a column-oriented database management system (DBMS) for online analytical processing of queries (OLAP).', 1, 3) AS Tuple;
```

Result:

```
Tuple
(("queries",'database','analytical'),('oriented','processing','DBMS')) |
```

## Functions for Generating Pseudo-Random Numbers

All the functions accept zero arguments or one argument. If an argument is passed, it can be any type, and its value is not used for anything. The only purpose of this argument is to prevent common subexpression elimination, so that two different instances of the same function return different columns with different random numbers.

### Note

Non-cryptographic generators of pseudo-random numbers are used.

## rand, rand32

Returns a pseudo-random UInt32 number, evenly distributed among all UInt32-type numbers.

Uses a linear congruential generator.

## rand64

Returns a pseudo-random UInt64 number, evenly distributed among all UInt64-type numbers.

Uses a linear congruential generator.

## randConstant

Produces a constant column with a random value.

### Syntax

```
randConstant([x])
```

### Arguments

- `x` — **Expression** resulting in any of the [supported data types](#). The resulting value is discarded, but the expression itself if used for bypassing [common subexpression elimination](#) if the function is called multiple times in one query. Optional parameter.

### Returned value

- Pseudo-random number.

Type: [UInt32](#).

### Example

Query:

```
SELECT rand(), rand(1), rand(number), randConstant(), randConstant(1), randConstant(number)
FROM numbers(3)
```

Result:

rand()	rand(1)	rand(number)	randConstant()	randConstant(1)	randConstant(number)
3047369878	4132449925	4044508545	2740811946	4229401477	1924032898
2938880146	1267722397	4154983056	2740811946	4229401477	1924032898
956619638	4238287282	1104342490	2740811946	4229401477	1924032898

## Random Functions for Working with Strings

`randomString`

`randomFixedString`

`randomPrintableASCII`

`randomStringUTF8`

`fuzzBits`

### Syntax

```
fuzzBits([s], [prob])
```

Inverts bits of `s`, each with probability `prob`.

### Arguments

- `s` - String or FixedString
- `prob` - constant Float32/64

### Returned value

Fuzzed string with same as `s` type.

### Example

```
SELECT fuzzBits(materialize('abacaba'), 0.1)
FROM numbers(3)
```

```
``` text
fuzzBits(materialize('abacaba'), 0.1)-
| abaaaja |
| a*cjab+ |
| aeca2A |
```

## Encoding Functions

`char`

Returns the string with the length as the number of passed arguments and each byte has the value of corresponding argument. Accepts multiple arguments of numeric types. If the value of argument is out of range of UInt8 data type, it is converted to UInt8 with possible rounding and overflow.

## Syntax

```
char(number_1, [number_2, ..., number_n]);
```

## Arguments

- `number_1, number_2, ..., number_n` — Numerical arguments interpreted as integers. Types: [Int](#), [Float](#).

## Returned value

- a string of given bytes.

Type: `String`.

## Example

Query:

```
SELECT char(104.1, 101, 108.9, 108.9, 111) AS hello;
```

Result:

```
hello  
hello |
```

You can construct a string of arbitrary encoding by passing the corresponding bytes. Here is example for UTF-8:

Query:

```
SELECT char(0xD0, 0xBF, 0xD1, 0x80, 0xD0, 0xB8, 0xD0, 0xB2, 0xD0, 0xB5, 0xD1, 0x82) AS hello;
```

Result:

```
hello  
привет |
```

Query:

```
SELECT char(0xE4, 0xBD, 0xA0, 0xE5, 0xA5, 0xBD) AS hello;
```

Result:

```
hello  
你好 |
```

hex

Returns a string containing the argument's hexadecimal representation.

Alias: `HEX`.

## Syntax

```
hex(arg)
```

The function is using uppercase letters A-F and not using any prefixes (like `0x`) or suffixes (like `h`).

For integer arguments, it prints hex digits ("nibbles") from the most significant to least significant (big-endian or "human-readable" order). It starts with the most significant non-zero byte (leading zero bytes are omitted) but always prints both digits of every byte even if the leading digit is zero.

Values of type `Date` and `DateTime` are formatted as corresponding integers (the number of days since Epoch for Date and the value of Unix Timestamp for DateTime).

For `String` and `FixedString`, all bytes are simply encoded as two hexadecimal numbers. Zero bytes are not omitted.

Values of `Float` and `Decimal` types are encoded as their representation in memory. As we support little-endian architecture, they are encoded in little-endian. Zero leading/trailing bytes are not omitted.

## Arguments

- `arg` — A value to convert to hexadecimal. Types: `String`, `UInt`, `Float`, `Decimal`, `Date` or `DateTime`.

## Returned value

- A string with the hexadecimal representation of the argument.

Type: `String`.

## Examples

Query:

```
SELECT hex(1);
```

Result:

```
01
```

Query:

```
SELECT hex(toFloat32(number)) AS hex_presentation FROM numbers(15, 2);
```

Result:

hex_presentation	00007041
	00008041

Query:

```
SELECT hex(toFloat64(number)) AS hex_presentation FROM numbers(15, 2);
```

Result:

```
hex_presentation
0000000000002E40
0000000000003040
```

## unhex

Performs the opposite operation of [hex](#). It interprets each pair of hexadecimal digits (in the argument) as a number and converts it to the byte represented by the number. The return value is a binary string (BLOB).

If you want to convert the result to a number, you can use the [reverse](#) and [reinterpretAs](#) functions.

### Note

If `unhex` is invoked from within the `clickhouse-client`, binary strings display using UTF-8.

Alias: UNHEX.

### Syntax

```
unhex(arg)
```

### Arguments

- arg — A string containing any number of hexadecimal digits. Type: [String](#).

Supports both uppercase and lowercase letters A-F. The number of hexadecimal digits does not have to be even. If it is odd, the last digit is interpreted as the least significant half of the 00-0F byte. If the argument string contains anything other than hexadecimal digits, some implementation-defined result is returned (an exception isn't thrown). For a numeric argument the inverse of `hex(N)` is not performed by `unhex()`.

### Returned value

- A binary string (BLOB).

Type: [String](#).

### Example

Query:

```
SELECT unhex('303132'), UNHEX('4D7953514C');
```

Result:

```
unhex('303132') unhex('4D7953514C')
012 | MySQL |
```

Query:

```
SELECT reinterpretAsUInt64(reverse(unhex('FFF'))) AS num;
```

Result:

```
num  
4095 |
```

## bin

Returns a string containing the argument's binary representation.

### Syntax

```
bin(arg)
```

Alias: `BIN`.

For integer arguments, it prints bin digits from the most significant to least significant (big-endian or "human-readable" order). It starts with the most significant non-zero byte (leading zero bytes are omitted) but always prints eight digits of every byte if the leading digit is zero.

Values of type `Date` and `DateTime` are formatted as corresponding integers (the number of days since Epoch for Date and the value of Unix Timestamp for DateTime).

For `String` and `FixedString`, all bytes are simply encoded as eight binary numbers. Zero bytes are not omitted.

Values of `Float` and `Decimal` types are encoded as their representation in memory. As we support little-endian architecture, they are encoded in little-endian. Zero leading/trailing bytes are not omitted.

### Arguments

- `arg` — A value to convert to binary. `String`, `FixedString`, `UInt`, `Float`, `Decimal`, `Date`, or `DateTime`.

### Returned value

- A string with the binary representation of the argument.

Type: `String`.

### Examples

Query:

```
SELECT bin(14);
```

Result:

```
bin(14)  
00001110 |
```

Query:

```
SELECT bin(toFloat32(number)) AS bin_presentation FROM numbers(15, 2);
```

## Result:

```
bin_presentation  
00000000000000000000111000001000001  
000000000000000000001000000001000001
```

## Query:

```
SELECT bin(toFloat64(number)) AS bin_presentation FROM numbers(15, 2);
```

## Result:

unbin

Interprets each pair of binary digits (in the argument) as a number and converts it to the byte represented by the number. The function performs the opposite operation to [bin](#).

## Syntax

unbin(arg)

Alias: UNBIN.

For a numeric argument `unbin()` does not return the inverse of `bin()`. If you want to convert the result to a number, you can use the `reverse` and `reinterpretAs` functions.

## Note

If unbin is invoked from within the clickhouse-client, binary strings are displayed using UTF-8.

Supports binary digits 0 and 1. The number of binary digits does not have to be multiples of eight. If the argument string contains anything other than binary digits, some implementation-defined result is returned (an exception isn't thrown).

## Arguments

- `arg` — A string containing any number of binary digits. `String`.

### Returned value

- A binary string (BLOB).

Type: String.

## Examples

Query:

```
SELECT UNBIN('001100000011000100110010'), UNBIN('010011010111001010100110101000101001100');
```

Result:

```
unbin('001100000011000100110010') | unbin('0100110101110010101001101000101001100')  
012 | MySQL |
```

Query:

```
SELECT reinterpretAsUInt64(reverse(unbin('1110'))) AS num;
```

Result:

```
num  
14 |
```

## UUIDStringToNum(str)

Accepts a string containing 36 characters in the format 123e4567-e89b-12d3-a456-426655440000, and returns it as a set of bytes in a FixedString(16).

## UUIDNumToString(str)

Accepts a FixedString(16) value. Returns a string containing 36 characters in text format.

## bitmaskToList(num)

Accepts an integer. Returns a string containing the list of powers of two that total the source number when summed. They are comma-separated without spaces in text format, in ascending order.

## bitmaskToArray(num)

Accepts an integer. Returns an array of UInt64 numbers containing the list of powers of two that total the source number when summed. Numbers in the array are in ascending order.

## bitPositionsToArray(num)

Accepts an integer and converts it to an unsigned integer. Returns an array of UInt64 numbers containing the list of positions of bits of arg that equal 1, in ascending order.

### Syntax

```
bitPositionsToArray(arg)
```

### Arguments

- arg — Integer value. [Int/UInt](#).

### Returned value

- An array containing a list of positions of bits that equal 1, in ascending order.

Type: [Array\(UInt64\)](#).

### Example

Query:

```
SELECT bitPositionsToArray(toInt8(1)) AS bit_positions;
```

Result:

```
bit_positions  
[0] |
```

Query:

```
SELECT bitPositionsToArray(toInt8(-1)) AS bit_positions;
```

Result:

```
bit_positions  
[0,1,2,3,4,5,6,7] |
```

## Functions for Working with UUID

The functions for working with UUID are listed below.

### generateUUIDv4

Generates the [UUID of version 4](#).

```
generateUUIDv4()
```

#### Returned value

The UUID type value.

#### Usage example

This example demonstrates creating a table with the UUID type column and inserting a value into the table.

```
CREATE TABLE t_uuid (x UUID) ENGINE=TinyLog  
INSERT INTO t_uuid SELECT generateUUIDv4()  
SELECT * FROM t_uuid
```

```
f4bf890f-f9dc-4332-ad5c-0c18e73f28e9 |
```

### empty

Checks whether the input UUID is empty.

#### Syntax

```
empty(UUID)
```

The UUID is considered empty if it contains all zeros (zero UUID).

The function also works for [arrays](#) or [strings](#).

## Arguments

- `x` — Input UUID. [UUID](#).

## Returned value

- Returns `1` for an empty UUID or `0` for a non-empty UUID.

Type: [UInt8](#).

## Example

To generate the UUID value, ClickHouse provides the [generateUUIDv4](#) function.

Query:

```
SELECT empty(generateUUIDv4());
```

Result:

```
+-----+  
| empty(generateUUIDv4()) |  
+-----+  
| 0 |
```

# notEmpty

Checks whether the input UUID is non-empty.

## Syntax

```
notEmpty(UUID)
```

The UUID is considered empty if it contains all zeros (zero UUID).

The function also works for [arrays](#) or [strings](#).

## Arguments

- `x` — Input UUID. [UUID](#).

## Returned value

- Returns `1` for a non-empty UUID or `0` for an empty UUID.

Type: [UInt8](#).

## Example

To generate the UUID value, ClickHouse provides the [generateUUIDv4](#) function.

Query:

```
SELECT notEmpty(generateUUIDv4());
```

Result:

```
notEmpty(generateUUIDv4())  
1 |
```

## toUUID (x)

Converts String type value to UUID type.

```
toUUID(String)
```

### Returned value

The UUID type value.

### Usage example

```
SELECT toUUID('61f0c404-5cb3-11e7-907b-a6006ad3dba0') AS uuid
```

```
61f0c404-5cb3-11e7-907b-a6006ad3dba0 |  
      ^uuid
```

## toUUIDOrNull (x)

It takes an argument of type String and tries to parse it into UUID. If failed, returns NULL.

```
toUUIDOrNull(String)
```

### Returned value

The Nullable(UUID) type value.

### Usage example

```
SELECT toUUIDOrNull('61f0c404-5cb3-11e7-907b-a6006ad3dba0T') AS uuid
```

```
uuid  
NULL |
```

## toUUIDOrZero (x)

It takes an argument of type String and tries to parse it into UUID. If failed, returns zero UUID.

```
toUUIDOrZero(String)
```

### Returned value

The UUID type value.

### Usage example

```
SELECT toUUIDOrZero('61f0c404-5cb3-11e7-907b-a6006ad3dba0T') AS uuid
```

00000000-0000-0000-0000-000000000000 | uuid

## UUIDStringToNum

Accepts a string containing 36 characters in the format `xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`, and returns it as a set of bytes in a [FixedString\(16\)](#).

```
UUIDStringToNum(String)
```

### Returned value

`FixedString(16)`

### Usage examples

```
SELECT
  '612f3c40-5d3b-217e-707b-6a546a3d7b29' AS uuid,
  UUIDStringToNum(uuid) AS bytes
```

612f3c40-5d3b-217e-707b-6a546a3d7b29 | bytes

## UUIDNumToString

Accepts a [FixedString\(16\)](#) value, and returns a string containing 36 characters in text format.

```
UUIDNumToString(FixedString(16))
```

### Returned value

`String`.

### Usage example

```
SELECT
  'a/<@];!~p{jTj={}' AS bytes,
  UUIDNumToString(toFixedString(bytes, 16)) AS uuid
```

a/<@];!~p{jTj={} | bytes

612f3c40-5d3b-217e-707b-6a546a3d7b29 | uuid

## serverUUID()

Returns the random and unique UUID, which is generated when the server is first started and stored forever. The result writes to the file `uuid` created in the ClickHouse server directory `/var/lib/clickhouse/`.

## Syntax

```
serverUUID()
```

### Returned value

- The UUID of the server.

Type: [UUID](#).

## See Also

- [dictGetUUID](#)

# Functions for Working with URLs

All these functions do not follow the RFC. They are maximally simplified for improved performance.

## Functions that Extract Parts of a URL

If the relevant part isn't present in a URL, an empty string is returned.

### protocol

Extracts the protocol from a URL.

Examples of typical returned values: http, https, ftp, mailto, tel, magnet...

### domain

Extracts the hostname from a URL.

```
domain(url)
```

### Arguments

- `url` — URL. Type: [String](#).

The URL can be specified with or without a scheme. Examples:

```
svn+ssh://some.svn-hosting.com:80/repo/trunk  
some.svn-hosting.com:80/repo/trunk  
https://yandex.com/time/
```

For these examples, the `domain` function returns the following results:

```
some.svn-hosting.com  
some.svn-hosting.com  
yandex.com
```

### Returned values

- Host name. If ClickHouse can parse the input string as a URL.

- Empty string. If ClickHouse can't parse the input string as a URL.

Type: `String`.

### Example

```
SELECT domain('svn+ssh://some.svn-hosting.com:80/repo/trunk');
```

```
domain('svn+ssh://some.svn-hosting.com:80/repo/trunk')—  
some.svn-hosting.com |
```

## domainWithoutWWW

Returns the domain and removes no more than one ‘www.’ from the beginning of it, if present.

### topLevelDomain

Extracts the the top-level domain from a URL.

```
topLevelDomain(url)
```

### Arguments

- `url` — URL. Type: `String`.

The URL can be specified with or without a scheme. Examples:

```
svn+ssh://some.svn-hosting.com:80/repo/trunk  
some.svn-hosting.com:80/repo/trunk  
https://yandex.com/time/
```

### Returned values

- Domain name. If ClickHouse can parse the input string as a URL.
- Empty string. If ClickHouse cannot parse the input string as a URL.

Type: `String`.

### Example

```
SELECT topLevelDomain('svn+ssh://www.some.svn-hosting.com:80/repo/trunk');
```

```
topLevelDomain('svn+ssh://www.some.svn-hosting.com:80/repo/trunk')—  
com |
```

## firstSignificantSubdomain

Returns the “first significant subdomain”. This is a non-standard concept specific to Yandex.Metrica. The first significant subdomain is a second-level domain if it is ‘com’, ‘net’, ‘org’, or ‘co’. Otherwise, it is a third-level domain. For example, `firstSignificantSubdomain ('https://news.yandex.ru/')` = ‘yandex’, `firstSignificantSubdomain ('https://news.yandex.com.tr/')` = ‘yandex’. The list of “insignificant” second-level domains and other implementation details may change in the future.

## cutToFirstSignificantSubdomain

Returns the part of the domain that includes top-level subdomains up to the “first significant subdomain” (see the explanation above).

For example:

- `cutToFirstSignificantSubdomain('https://news.yandex.com.tr/')` = 'yandex.com.tr'.
- `cutToFirstSignificantSubdomain('www.tr')` = 'tr'.
- `cutToFirstSignificantSubdomain('tr')` = ''.

## cutToFirstSignificantSubdomainWithWWW

Returns the part of the domain that includes top-level subdomains up to the “first significant subdomain”, without stripping “www”.

For example:

- `cutToFirstSignificantSubdomain('https://news.yandex.com.tr/')` = 'yandex.com.tr'.
- `cutToFirstSignificantSubdomain('www.tr')` = 'www.tr'.
- `cutToFirstSignificantSubdomain('tr')` = ''.

## cutToFirstSignificantSubdomainCustom

Returns the part of the domain that includes top-level subdomains up to the first significant subdomain. Accepts custom **TLD list** name.

Can be useful if you need fresh TLD list or you have custom.

Configuration example:

```
<!-- <top_level_domains_path>/var/lib/clickhouse/top_level_domains/</top_level_domains_path> -->
<top_level_domains_lists>
    <!-- https://publicsuffix.org/list/public_suffix_list.dat -->
    <public_suffix_list>public_suffix_list.dat</public_suffix_list>
    <!-- NOTE: path is under top_level_domains_path -->
</top_level_domains_lists>
```

## Syntax

```
cutToFirstSignificantSubdomain(URL, TLD)
```

## Parameters

- `URL` — URL. **String**.
- `TLD` — Custom TLD list name. **String**.

## Returned value

- Part of the domain that includes top-level subdomains up to the first significant subdomain.

Type: **String**.

## Example

Query:

```
SELECT cutToFirstSignificantSubdomainCustom('bar.foo.there-is-no-such-domain', 'public_suffix_list');
```

Result:

```
cutToFirstSignificantSubdomainCustom('bar.foo.there-is-no-such-domain', 'public_suffix_list')  
|  
foo.there-is-no-such-domain
```

## See Also

- [firstSignificantSubdomain](#).

## cutToFirstSignificantSubdomainCustomWithWWW

Returns the part of the domain that includes top-level subdomains up to the first significant subdomain without stripping `www`. Accepts custom TLD list name.

Can be useful if you need fresh TLD list or you have custom.

Configuration example:

```
<!-- <top_level_domains_path>/var/lib/clickhouse/top_level_domains/</top_level_domains_path> -->  
<top_level_domains_lists>  
  <!-- https://publicsuffix.org/list/public_suffix_list.dat -->  
  <public_suffix_list>public_suffix_list.dat</public_suffix_list>  
  <!-- NOTE: path is under top_level_domains_path -->  
</top_level_domains_lists>
```

## Syntax

```
cutToFirstSignificantSubdomainCustomWithWWW(URL, TLD)
```

## Parameters

- `URL` — URL. [String](#).
- `TLD` — Custom TLD list name. [String](#).

## Returned value

- Part of the domain that includes top-level subdomains up to the first significant subdomain without stripping `www`.

Type: [String](#).

## Example

Query:

```
SELECT cutToFirstSignificantSubdomainCustomWithWWW('www.foo', 'public_suffix_list');
```

Result:

```
cutToFirstSignificantSubdomainCustomWithWWW('www.foo', 'public_suffix_list')  
|  
www.foo
```

## See Also

- [firstSignificantSubdomain](#).

## firstSignificantSubdomainCustom

Returns the first significant subdomain. Accepts customs TLD list name.

Can be useful if you need fresh TLD list or you have custom.

Configuration example:

```
<!-- <top_level_domains_path>/var/lib/clickhouse/top_level_domains/</top_level_domains_path> -->
<top_level_domains_lists>
    <!-- https://publicsuffix.org/list/public_suffix_list.dat -->
    <public_suffix_list>public_suffix_list.dat</public_suffix_list>
    <!-- NOTE: path is under top_level_domains_path -->
</top_level_domains_lists>
```

## Syntax

```
firstSignificantSubdomainCustom(URL, TLD)
```

## Parameters

- URL — URL. [String](#).
- TLD — Custom TLD list name. [String](#).

## Returned value

- First significant subdomain.

Type: [String](#).

## Example

Query:

```
SELECT firstSignificantSubdomainCustom('bar.foo.there-is-no-such-domain', 'public_suffix_list');
```

Result:

```
firstSignificantSubdomainCustom('bar.foo.there-is-no-such-domain', 'public_suffix_list')—  
| foo
```

## See Also

- [firstSignificantSubdomain](#).

## port(URL[, default\_port = 0])

Returns the port or `default_port` if there is no port in the URL (or in case of validation error).

## path

Returns the path. Example: `/top/news.html` The path does not include the query string.

## pathFull

The same as above, but including query string and fragment. Example: /top/news.html?page=2#comments

## queryString

Returns the query string. Example: page=1&lr=213. query-string does not include the initial question mark, as well as # and everything after #.

## fragment

Returns the fragment identifier. fragment does not include the initial hash symbol.

## queryStringAndFragment

Returns the query string and fragment identifier. Example: page=1#29390.

## extractURLParameter(URL, name)

Returns the value of the 'name' parameter in the URL, if present. Otherwise, an empty string. If there are many parameters with this name, it returns the first occurrence. This function works under the assumption that the parameter name is encoded in the URL exactly the same way as in the passed argument.

## extractURLParameters(URL)

Returns an array of name=value strings corresponding to the URL parameters. The values are not decoded in any way.

## extractURLParameterNames(URL)

Returns an array of name strings corresponding to the names of URL parameters. The values are not decoded in any way.

## URLHierarchy(URL)

Returns an array containing the URL, truncated at the end by the symbols /,? in the path and query-string. Consecutive separator characters are counted as one. The cut is made in the position after all the consecutive separator characters.

## URLPathHierarchy(URL)

The same as above, but without the protocol and host in the result. The / element (root) is not included. Example: the function is used to implement tree reports the URL in Yandex. Metric.

```
URLPathHierarchy('https://example.com/browse/CONV-6788') =  
[  
  '/browse/',  
  '/browse/CONV-6788'  
]
```

## decodeURLComponent(URL)

Returns the decoded URL.

Example:

```
SELECT decodeURLComponent('http://127.0.0.1:8123/?query=SELECT%201%3B') AS DecodedURL;
```

```
DecodedURL—  
http://127.0.0.1:8123/?query=SELECT 1; |
```

## netloc

Extracts network locality (username:password@host:port) from a URL.

### Syntax

```
netloc(URL)
```

### Arguments

- `url` — URL. **String**.

### Returned value

- `username:password@host:port`.

Type: **String**.

### Example

Query:

```
SELECT netloc('http://paul@www.example.com:80/');
```

Result:

```
netloc('http://paul@www.example.com:80/')—  
paul@www.example.com:80 |
```

## Functions that Remove Part of a URL

If the URL does not have anything similar, the URL remains unchanged.

### cutWWW

Removes no more than one ‘www.’ from the beginning of the URL’s domain, if present.

### cutQueryString

Removes query string. The question mark is also removed.

### cutFragment

Removes the fragment identifier. The number sign is also removed.

### cutQueryStringAndFragment

Removes the query string and fragment identifier. The question mark and number sign are also removed.

### cutURLParameter(URL, name)

Removes the ‘name’ URL parameter, if present. This function works under the assumption that the parameter name is encoded in the URL exactly the same way as in the passed argument.

## Functions for Working with IPv4 and IPv6 Addresses

## IPv4NumToString(num)

Takes a UInt32 number. Interprets it as an IPv4 address in big endian. Returns a string containing the corresponding IPv4 address in the format A.B.C.d (dot-separated numbers in decimal form).

Alias: INET\_NTOA.

## IPv4StringToNum(s)

The reverse function of IPv4NumToString. If the IPv4 address has an invalid format, it returns 0.

Alias: INET\_ATON.

**IPv4NumToStringClassC(num)**

Similar to IPv4NumToString, but using xxx instead of the last octet.

## Example:

```
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

Since using 'xxx' is highly unusual, this may be changed in the future. We recommend that you do not rely on the exact format of this fragment.

## IPv6NumToString(x)

Accepts a FixedString(16) value containing the IPv6 address in binary format. Returns a string containing this address in text format.

IPv6-mapped IPv4 addresses are output in the format ::ffff:111.222.33.44.

Alias: INET6\_NTOA

## Examples:

```
SELECT IPv6NumToString(toFixedString(unhex('2A0206B800000000000000000000000011'), 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('00000000000000000000FFFF')  
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

The reverse function of [IPv6NumToString](#). If the IPv6 address has an invalid format, it returns a string of null bytes.

If the input string contains a valid IPv4 address, returns its IPv6 equivalent.  
HEX can be uppercase or lowercase.

Alias: INET6\_ATON.

### Syntax

```
IPv6StringToNum(string)
```

## Argument

- `string` — IP address. [String](#).

## Returned value

- IPv6 address in binary format.

Type: [FixedString\(16\)](#).

## Example

Query:

```
SELECT addr, cutIPv6(IPv6StringToNum(addr), 0, 0) FROM (SELECT ['notaddress', '127.0.0.1', '1111::ffff'] AS addr)
ARRAY JOIN addr;
```

Result:

addr	cutIPv6(IPv6StringToNum(addr), 0, 0)
notaddress   ::	
127.0.0.1   ::ffff:127.0.0.1	
1111::ffff   1111::ffff	

## See Also

- [cutIPv6](#).

## IPv4ToIPv6(x)

Takes a `UInt32` number. Interprets it as an IPv4 address in [big endian](#). Returns a `FixedString(16)` value containing the IPv6 address in binary format. Examples:

```
SELECT IPv6NumToString(IPv4ToIPv6(IPv4StringToNum('192.168.0.1'))) AS addr;
```

addr
::ffff:192.168.0.1

## cutIPv6(x, bytesToCutForIPv6, bytesToCutForIPv4)

Accepts a `FixedString(16)` value containing the IPv6 address in binary format. Returns a string containing the address of the specified number of bytes removed in text format. For example:

```
WITH
    IPv6StringToNum('2001:0DB8:AC10:FE01:FEED:BABE:CAFE:F00D') AS ipv6,
    IPv4ToIPv6(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, Cidr),

Accepts an IPv4 and an UInt8 value containing the CIDR. Return a tuple with two IPv4 containing the lower range and the higher range of the subnet.

```
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, Cidr),

Accepts an IPv6 and an UInt8 value containing the CIDR. Return a tuple with two IPv6 containing the lower range and the higher range of the subnet.

```
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:ffff') ┌─┘
```

## toIPv4(string)

An alias to [IPv4StringToNum\(\)](#) that takes a string form of IPv4 address and returns value of [IPv4](#) type, which is binary equal to value returned by [IPv4StringToNum\(\)](#).

```
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

Converts a string form of IPv6 address to [IPv6](#) type. If the IPv6 address has an invalid format, returns an empty value.

Similar to [IPv6StringToNum](#) function, which converts IPv6 address to binary format.

If the input string contains a valid IPv4 address, then the IPv6 equivalent of the IPv4 address is returned.

## Syntax

```
toIPv6(string)
```

## Argument

- string — IP address. **String**.

## Returned value

- IP address.

Type: **IPv6**.

## Examples

Query:

```
WITH '2001:438:ffff::407d:1bc1' AS IPv6_string
SELECT
    hex(IPv6StringToNum(IPv6_string)),
    hex(toIPv6(IPv6_string));
```

Result:

```
hex(IPv6StringToNum(IPv6_string)) | hex(toIPv6(IPv6_string))
20010438FFFF00000000000407D1BC1 | 20010438FFFF00000000000407D1BC1 |
```

Query:

```
SELECT toIPv6('127.0.0.1');
```

Result:

```
toIPv6('127.0.0.1') |
::ffff:127.0.0.1 |
```

## isIPv4String

Determines whether the input string is an IPv4 address or not. If `string` is IPv6 address returns 0.

## Syntax

```
isIPv4String(string)
```

## Arguments

- string — IP address. **String**.

## Returned value

- 1 if `string` is IPv4 address, 0 otherwise.

Type: **UInt8**.

## Examples

Query:

```
SELECT addr, isIPv4String(addr) FROM ( SELECT ['0.0.0.0', '127.0.0.1', '::ffff:127.0.0.1'] AS addr ) ARRAY JOIN addr;
```

Result:

addr	isIPv4String(addr)
0.0.0.0	1
127.0.0.1	1
::ffff:127.0.0.1	0

## isIPv6String

Determines whether the input string is an IPv6 address or not. If `string` is IPv4 address returns 0.

### Syntax

```
isIPv6String(string)
```

### Arguments

- `string` — IP address. [String](#).

### Returned value

- 1 if `string` is IPv6 address, 0 otherwise.

Type: [UInt8](#).

### Examples

Query:

```
SELECT addr, isIPv6String(addr) FROM ( SELECT ['::', '1111::ffff', '::ffff:127.0.0.1', '127.0.0.1'] AS addr ) ARRAY JOIN addr;
```

Result:

addr	isIPv6String(addr)
::	1
1111::ffff	1
::ffff:127.0.0.1	1
127.0.0.1	0

## isIPAddressInRange

Determines if an IP address is contained in a network represented in the [CIDR](#) notation. Returns 1 if true, or 0 otherwise.

### Syntax

```
isIPAddressInRange(address, prefix)
```

This function accepts both IPv4 and IPv6 addresses (and networks) represented as strings. It returns 0 if the IP version of the address and the CIDR don't match.

## Arguments

- `address` — An IPv4 or IPv6 address. [String](#).
- `prefix` — An IPv4 or IPv6 network prefix in CIDR. [String](#).

## Returned value

- 1 or 0.

Type: [UInt8](#).

## Example

Query:

```
SELECT isIPAddressInRange('127.0.0.1', '127.0.0.0/8');
```

Result:

```
isIPAddressInRange('127.0.0.1', '127.0.0.0/8')  
1 |
```

Query:

```
SELECT isIPAddressInRange('127.0.0.1', 'ffff::/16');
```

Result:

```
isIPAddressInRange('127.0.0.1', 'ffff::/16')  
0 |
```

# Functions for Working with JSON

In Yandex.Metrica, JSON is transmitted by users as session parameters. There are some special functions for working with this JSON. (Although in most of the cases, the JSONs are additionally pre-processed, and the resulting values are put in separate columns in their processed format.) All these functions are based on strong assumptions about what the JSON can be, but they try to do as little as possible to get the job done.

The following assumptions are made:

1. The field name (function argument) must be a constant.
2. The field name is somehow canonically encoded in JSON. For example: `visitParamHas('{"abc":"def"}', 'abc')` = 1, but `visitParamHas('"\u0061\u0062\u0063":"def"', 'abc')` = 0
3. Fields are searched for on any nesting level, indiscriminately. If there are multiple matching fields, the first occurrence is used.
4. The JSON does not have space characters outside of string literals.

## visitParamHas(params, name)

Checks whether there is a field with the `name` name.

Alias: `simpleJSONHas`.

## visitParamExtractUInt(params, name)

Parses UInt64 from the value of the field named `name`. If this is a string field, it tries to parse a number from the beginning of the string. If the field does not exist, or it exists but does not contain a number, it returns 0.

Alias: `simpleJSONExtractUInt`.

## visitParamExtractInt(params, name)

The same as for Int64.

Alias: `simpleJSONExtractInt`.

## visitParamExtractFloat(params, name)

The same as for Float64.

Alias: `simpleJSONExtractFloat`.

## visitParamExtractBool(params, name)

Parses a true/false value. The result is UInt8.

Alias: `simpleJSONExtractBool`.

## visitParamExtractRaw(params, name)

Returns the value of a field, including separators.

Alias: `simpleJSONExtractRaw`.

Examples:

```
visitParamExtractRaw('{"abc":"\\n\\u0000"}', 'abc') = "\\n\\u0000";
visitParamExtractRaw('{"abc":{"def":[1,2,3]}}', 'abc') = '{"def":[1,2,3]}';
```

## visitParamExtractString(params, name)

Parses the string in double quotes. The value is unescaped. If unescaping failed, it returns an empty string.

Alias: `simpleJSONExtractString`.

Examples:

```
visitParamExtractString('{"abc":"\\n\\u0000"}', 'abc') = '\\n\\0';
visitParamExtractString('{"abc":"\\u263a"}', 'abc') = '@';
visitParamExtractString('{"abc":"\\u263"}', 'abc') = '';
visitParamExtractString('{"abc":"hello"}', 'abc') = '';
```

There is currently no support for code points in the format `\uXXXX\uYYYY` that are not from the basic multilingual plane (they are converted to CESU-8 instead of UTF-8).

The following functions are based on `simdjson` designed for more complex JSON parsing requirements. The assumption 2 mentioned above still applies.

## isValidJSON(json)

Checks that passed string is a valid json.

Examples:

```
SELECT isValidJSON('{"a": "hello", "b": [-100, 200.0, 300]}') = 1
SELECT isValidJSON('not a json') = 0
```

## JSONHas(json[, indices\_or\_keys]...)

If the value exists in the JSON document, 1 will be returned.

If the value does not exist, 0 will be returned.

Examples:

```
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` is a list of zero or more arguments each of them can be either string or integer.

- String = access object member by key.
- Positive integer = access the n-th member/key from the beginning.
- Negative integer = access the n-th member/key from the end.

Minimum index of the element is 1. Thus the element 0 does not exist.

You may use integers to access both JSON arrays and JSON objects.

So, for example:

```
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]...)

Return the length of a JSON array or a JSON object.

If the value does not exist or has a wrong type, 0 will be returned.

Examples:

```
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]...)

Return the type of a JSON value.

If the value does not exist, Null will be returned.

Examples:

```
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]...)**

Parses a JSON and extract a value. These functions are similar to `visitParam` functions.

If the value does not exist or has a wrong type, 0 will be returned.

Examples:

```
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]...)**

Parses a JSON and extract a string. This function is similar to `visitParamExtractString` functions.

If the value does not exist or has a wrong type, an empty string will be returned.

The value is unescaped. If unescaping failed, it returns an empty string.

Examples:

```
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": "\u263"}', 'abc') = "
SELECT JSONExtractString('{"abc": "hello"}', 'abc') = "
```

**JSONExtract(json[, indices\_or\_keys...], Return\_type)**

Parses a JSON and extract a value of the given ClickHouse data type.

This is a generalization of the previous `JSONExtract<type>` functions.

This means

`JSONExtract(..., 'String')` returns exactly the same as `JSONExtractString()`,

`JSONExtract(..., 'Float64')` returns exactly the same as `JSONExtractFloat()`.

Examples:

```
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)**

Parses key-value pairs from a JSON where the values are of the given ClickHouse data type.

Example:

```
SELECT JSONExtractKeysAndValues('{"x": {"a": 5, "b": 7, "c": 11}}', 'x', 'Int8') = [('a',5),('b',7),('c',11)];
```

## JSONExtractRaw(json[, indices\_or\_keys]...)

Returns a part of JSON as unparsed string.

If the part does not exist or has a wrong type, an empty string will be returned.

Example:

```
SELECT JSONExtractRaw('{"a": "hello", "b": [-100, 200.0, 300]}', 'b') = '[-100, 200.0, 300];
```

## JSONExtractArrayRaw(json[, indices\_or\_keys...] )

Returns an array with elements of JSON array, each represented as unparsed string.

If the part does not exist or isn't array, an empty array will be returned.

Example:

```
SELECT JSONExtractArrayRaw('{"a": "hello", "b": [-100, 200.0, "hello"]}', 'b') = ['-100', '200.0', '"hello"]];
```

## JSONExtractKeysAndValuesRaw

Extracts raw data from a JSON object.

### Syntax

```
JSONExtractKeysAndValuesRaw(json[, p, a, t, h])
```

### Arguments

- **json** — **String** with valid JSON.
- **p, a, t, h** — Comma-separated indices or keys that specify the path to the inner field in a nested JSON object. Each argument can be either a **string** to get the field by the key or an **integer** to get the N-th field (indexed from 1, negative integers count from the end). If not set, the whole JSON is parsed as the top-level object. Optional parameter.

### Returned values

- Array with ('key', 'value') tuples. Both tuple members are strings.
- Empty array if the requested object does not exist, or input JSON is invalid.

Type: **Array(Tuple(String, String))**.

### Examples

Query:

```
SELECT JSONExtractKeysAndValuesRaw('{"a": [-100, 200.0], "b":{"c": {"d": "hello", "f": "world"} }}');
```

Result:

```
JSONExtractKeysAndValuesRaw('{"a": [-100, 200.0], "b": {"c": {"d": "hello", "f": "world"} }}')-->[('a', '[-100,200]'), ('b', {'c': {'d': 'hello', 'f': 'world'}})]
```

Query:

```
SELECT JSONExtractKeysAndValuesRaw('{"a": [-100, 200.0], "b": {"c": {"d": "hello", "f": "world"} }}', 'b');
```

Result:

```
JSONExtractKeysAndValuesRaw('{"a": [-100, 200.0], "b": {"c": {"d": "hello", "f": "world"} }}', 'b')-->[('c', {'d': 'hello', 'f': 'world'})]
```

Query:

```
SELECT JSONExtractKeysAndValuesRaw('{"a": [-100, 200.0], "b": {"c": {"d": "hello", "f": "world"} }}', -1, 'c');
```

Result:

```
JSONExtractKeysAndValuesRaw('{"a": [-100, 200.0], "b": {"c": {"d": "hello", "f": "world"} }}', -1, 'c')-->[('d', 'hello'), ('f', 'world')]
```

## JSON\_EXISTS(json, path)

If the value exists in the JSON document, 1 will be returned.

If the value does not exist, 0 will be returned.

Examples:

```
SELECT JSON_EXISTS('{"hello":1}', '$.hello');  
SELECT JSON_EXISTS('{"hello":{"world":1}}', '$.hello.world');  
SELECT JSON_EXISTS('{"hello":["world"]}', '$.hello[*]');  
SELECT JSON_EXISTS('{"hello":["world"]}', '$.hello[0]');
```

### Note

before version 21.11 the order of arguments was wrong, i.e. JSON\_EXISTS(path, json)

## JSON\_QUERY(json, path)

Parses a JSON and extract a value as JSON array or JSON object.

If the value does not exist, an empty string will be returned.

Example:

```
SELECT JSON_QUERY('{"hello":"world"}', '$.hello');
SELECT JSON_QUERY('{"array":[[0, 1, 2, 3, 4, 5], [0, -1, -2, -3, -4, -5]]}', '$.array[*][0 to 2, 4]');
SELECT JSON_QUERY('{"hello":2}', '$.hello');
SELECT typeName(JSON_QUERY('{"hello":2}', '$.hello'));
```

Result:

```
["world"]
[0, 1, 4, 0, -1, -4]
[2]
String
```

## Note

before version 21.11 the order of arguments was wrong, i.e. `JSON_QUERY(path, json)`

## JSON\_VALUE(json, path)

Parses a JSON and extract a value as JSON scalar.

If the value does not exist, an empty string will be returned.

Example:

```
SELECT JSON_VALUE('{"hello":"world"}', '$.hello');
SELECT JSON_VALUE('{"array":[[0, 1, 2, 3, 4, 5], [0, -1, -2, -3, -4, -5]]}', '$.array[*][0 to 2, 4]');
SELECT JSON_VALUE('{"hello":2}', '$.hello');
SELECT typeName(JSON_VALUE('{"hello":2}', '$.hello'));
```

Result:

```
"world"
0
2
String
```

## Note

before version 21.11 the order of arguments was wrong, i.e. `JSON_VALUE(path, json)`

## toJSONString

Serializes a value to its JSON representation. Various data types and nested structures are supported.

64-bit [integers](#) or bigger (like `UInt64` or `Int128`) are enclosed in quotes by default.

`output_format_json_quote_64bit_integers` controls this behavior.

Special values `Nan` and `inf` are replaced with `null`. Enable `output_format_json_quote_denormals` setting to show them.

When serializing an [Enum](#) value, the function outputs its name.

### Syntax

```
toJSONString(value)
```

## Arguments

- `value` — Value to serialize. Value may be of any data type.

## Returned value

- JSON representation of the value.

Type: [String](#).

## Example

The first example shows serialization of a [Map](#).

The second example shows some special values wrapped into a [Tuple](#).

Query:

```
SELECT toJSONString(map('key1', 1, 'key2', 2));
SELECT toJSONString(tuple(1.25, NULL, NaN, +inf, -inf, [])) SETTINGS output_format_json_quote_denormals = 1;
```

Result:

```
{"key1":1,"key2":2}
[1.25,null,"nan","inf","-inf",[]]
```

## See Also

- [output\\_format\\_json\\_quote\\_64bit\\_integers](#)
- [output\\_format\\_json\\_quote\\_denormals](#)

## Attention

For dictionaries, created with [DDL queries](#), the `dict_name` parameter must be fully specified, like `<database>.⟨dict_name⟩`. Otherwise, the current database is used.

# Functions for Working with External Dictionaries

For information on connecting and configuring external dictionaries, see [External dictionaries](#).

## dictGet, dictGetOrDefault, dictGetOrNull

Retrieves values from an external dictionary.

```
dictGet('dict_name', attr_names, id_expr)
dictGetOrDefault('dict_name', attr_names, id_expr, default_value_expr)
dictGetOrNull('dict_name', attr_name, id_expr)
```

## Arguments

- `dict_name` — Name of the dictionary. [String literal](#).
- `attr_names` — Name of the column of the dictionary, [String literal](#), or tuple of column names, [Tuple\(String literal\)](#).

- `id_expr` — Key value. [Expression](#) returning dictionary key-type value or [Tuple](#)-type value depending on the dictionary configuration.
- `default_value_expr` — Values returned if the dictionary does not contain a row with the `id_expr` key. [Expression](#) or [Tuple\(Expression\)](#), returning the value (or values) in the data types configured for the `attr_names` attribute.

## Returned value

- If ClickHouse parses the attribute successfully in the [attribute's data type](#), functions return the value of the dictionary attribute that corresponds to `id_expr`.
- If there is no the key, corresponding to `id_expr`, in the dictionary, then:

- `dictGet` returns the content of the `<null\_value>` element specified for the attribute in the dictionary configuration.  
 - `dictGetOrDefault` returns the value passed as the `default_value_expr` parameter.  
 - `dictGetOrNull` returns `'NULL'` in case key was not found in dictionary.

ClickHouse throws an exception if it cannot parse the value of the attribute or the value does not match the attribute data type.

## Example for simple key dictionary

Create a text file `ext-dict-test.csv` containing the following:

```
1,1
2,2
```

The first column is `id`, the second column is `c1`.

Configure the external dictionary:

```
<clickhouse>
  <dictionary>
    <name>ext-dict-test</name>
    <source>
      <file>
        <path>/path-to/ext-dict-test.csv</path>
        <format>CSV</format>
      </file>
    </source>
    <layout>
      <flat />
    </layout>
    <structure>
      <id>
        <name>id</name>
      </id>
      <attribute>
        <name>c1</name>
        <type>UInt32</type>
        <null_value></null_value>
      </attribute>
    </structure>
    <lifetime>0</lifetime>
  </dictionary>
</clickhouse>
```

Perform the query:

```

SELECT
    dictGetOrDefault('ext-dict-test', 'c1', number + 1, toUInt32(number * 10)) AS val,
    toTypeName(val) AS type
FROM system.numbers
LIMIT 3;

```

val	type
1	UInt32
2	UInt32
20	UInt32

## Example for complex key dictionary

Create a text file `ext-dict-mult.csv` containing the following:

```

1,1,'1'
2,2,'2'
3,3,'3'

```

The first column is `id`, the second is `c1`, the third is `c2`.

Configure the external dictionary:

```

<clickhouse>
    <dictionary>
        <name>ext-dict-mult</name>
        <source>
            <file>
                <path>/path-to/ext-dict-mult.csv</path>
                <format>CSV</format>
            </file>
        </source>
        <layout>
            <flat />
        </layout>
        <structure>
            <id>
                <name>id</name>
            </id>
            <attribute>
                <name>c1</name>
                <type>UInt32</type>
                <null_value></null_value>
            </attribute>
            <attribute>
                <name>c2</name>
                <type>String</type>
                <null_value></null_value>
            </attribute>
        </structure>
        <lifetime>0</lifetime>
    </dictionary>
</clickhouse>

```

Perform the query:

```

SELECT
    dictGet('ext-dict-mult', ('c1','c2'), number) AS val,
    toTypeName(val) AS type
FROM system.numbers
LIMIT 3;

```

val	type
(1,'1')	Tuple(UInt8, String)
(2,'2')	Tuple(UInt8, String)
(3,'3')	Tuple(UInt8, String)

## Example for range key dictionary

Input table:

```
CREATE TABLE range_key_dictionary_source_table
(
    key UInt64,
    start_date Date,
    end_date Date,
    value String,
    value_nullable Nullable(String)
)
ENGINE = TinyLog();

INSERT INTO range_key_dictionary_source_table VALUES(1, toDate('2019-05-20'), toDate('2019-05-20'), 'First', 'First');
INSERT INTO range_key_dictionary_source_table VALUES(2, toDate('2019-05-20'), toDate('2019-05-20'), 'Second',
NULL);
INSERT INTO range_key_dictionary_source_table VALUES(3, toDate('2019-05-20'), toDate('2019-05-20'), 'Third',
'Third');
```

Create the external dictionary:

```
CREATE DICTIONARY range_key_dictionary
(
    key UInt64,
    start_date Date,
    end_date Date,
    value String,
    value_nullable Nullable(String)
)
PRIMARY KEY key
SOURCE(CLICKHOUSE(HOST 'localhost' PORT tcpPort() TABLE 'range_key_dictionary_source_table'))
LIFETIME(MIN 1 MAX 1000)
LAYOUT(RANGE_HASHED())
RANGE(MIN start_date MAX end_date);
```

Perform the query:

```
SELECT
    (number, toDate('2019-05-20')),
    dictHas('range_key_dictionary', number, toDate('2019-05-20')),
    dictGetOrNull('range_key_dictionary', 'value', number, toDate('2019-05-20')),
    dictGetOrNull('range_key_dictionary', 'value_nullable', number, toDate('2019-05-20')),
    dictGetOrNull('range_key_dictionary', ('value', 'value_nullable'), number, toDate('2019-05-20'))
FROM system.numbers LIMIT 5 FORMAT TabSeparated;
```

Result:

(0,'2019-05-20')	0	\N	\N	(NULL,NULL)
(1,'2019-05-20')	1	First	First	('First','First')
(2,'2019-05-20')	0	\N	\N	(NULL,NULL)
(3,'2019-05-20')	0	\N	\N	(NULL,NULL)
(4,'2019-05-20')	0	\N	\N	(NULL,NULL)

## See Also

- [External Dictionaries](#)

# dictHas

Checks whether a key is present in a dictionary.

```
dictHas('dict_name', id_expr)
```

## Arguments

- `dict_name` — Name of the dictionary. [String literal](#).
- `id_expr` — Key value. [Expression](#) returning dictionary key-type value or [Tuple](#)-type value depending on the dictionary configuration.

## Returned value

- 0, if there is no key.
- 1, if there is a key.

Type: [UInt8](#).

# dictGetHierarchy

Creates an array, containing all the parents of a key in the [hierarchical dictionary](#).

## Syntax

```
dictGetHierarchy('dict_name', key)
```

## Arguments

- `dict_name` — Name of the dictionary. [String literal](#).
- `key` — Key value. [Expression](#) returning a [UInt64](#)-type value.

## Returned value

- Parents for the key.

Type: [Array\(UInt64\)](#).

# dictIsIn

Checks the ancestor of a key through the whole hierarchical chain in the dictionary.

```
dictIsIn('dict_name', child_id_expr, ancestor_id_expr)
```

## Arguments

- `dict_name` — Name of the dictionary. [String literal](#).
- `child_id_expr` — Key to be checked. [Expression](#) returning a [UInt64](#)-type value.
- `ancestor_id_expr` — Alleged ancestor of the `child_id_expr` key. [Expression](#) returning a [UInt64](#)-type value.

## Returned value

- 0, if `child_id_expr` is not a child of `ancestor_id_expr`.

- 1, if `child_id_expr` is a child of `ancestor_id_expr` or if `child_id_expr` is an `ancestor_id_expr`.

Type: `UInt8`.

## dictGetChildren

Returns first-level children as an array of indexes. It is the inverse transformation for [dictGetHierarchy](#).

### Syntax

```
dictGetChildren(dict_name, key)
```

### Arguments

- `dict_name` — Name of the dictionary. [String literal](#).
- `key` — Key value. [Expression](#) returning a `UInt64`-type value.

### Returned values

- First-level descendants for the key.

Type: [Array\(UInt64\)](#).

### Example

Consider the hierachic dictionary:

id	parent_id
1	0
2	1
3	1
4	2

First-level children:

```
SELECT dictGetChildren('hierarchy_flat_dictionary', number) FROM system.numbers LIMIT 4;
```

dictGetChildren('hierarchy_flat_dictionary', number) ↴	
[1]	
[2,3]	
[4]	
[]	

## dictGetDescendant

Returns all descendants as if [dictGetChildren](#) function was applied `level` times recursively.

### Syntax

```
dictGetDescendants(dict_name, key, level)
```

### Arguments

- `dict_name` — Name of the dictionary. [String literal](#).
- `key` — Key value. [Expression](#) returning a `UInt64`-type value.

- `level` — Hierarchy level. If `level = 0` returns all descendants to the end. **UInt8**.

## Returned values

- Descendants for the key.

Type: **Array(UInt64)**.

## Example

Consider the hierachic dictionary:

id	parent_id
1	0
2	1
3	1
4	2

All descendants:

```
SELECT dictGetDescendants('hierarchy_flat_dictionary', number) FROM system.numbers LIMIT 4;
```

```
dictGetDescendants('hierarchy_flat_dictionary', number)→
[1,2,3,4]   |
[2,3,4]   |
[4]   |
[]   |
```

First-level descendants:

```
SELECT dictGetDescendants('hierarchy_flat_dictionary', number, 1) FROM system.numbers LIMIT 4;
```

```
dictGetDescendants('hierarchy_flat_dictionary', number, 1)→
[1]   |
[2,3]   |
[4]   |
[]   |
```

## Other Functions

ClickHouse supports specialized functions that convert dictionary attribute values to a specific data type regardless of the dictionary configuration.

Functions:

- `dictGetInt8`, `dictGetInt16`, `dictGetInt32`, `dictGetInt64`
- `dictGetUInt8`, `dictGetUInt16`, `dictGetUInt32`, `dictGetUInt64`
- `dictGetFloat32`, `dictGetFloat64`
- `dictGetDate`
- `dictGetDateTime`
- `dictGetUUID`

- `dictGetString`

All these functions have the `OrDefault` modification. For example, `dictGetDateOrDefault`.

Syntax:

```
dictGet[Type](#sql-reference-functions--dict_name---attr_name---id_expr)
dictGet[Type]OrDefault('dict_name', 'attr_name', id_expr, default_value_expr)
```

## Arguments

- `dict_name` — Name of the dictionary. [String literal](#).
- `attr_name` — Name of the column of the dictionary. [String literal](#).
- `id_expr` — Key value. [Expression](#) returning a `UInt64` or `Tuple`-type value depending on the dictionary configuration.
- `default_value_expr` — Value returned if the dictionary does not contain a row with the `id_expr` key. [Expression](#) returning the value in the data type configured for the `attr_name` attribute.

## Returned value

- If ClickHouse parses the attribute successfully in the [attribute's data type](#), functions return the value of the dictionary attribute that corresponds to `id_expr`.
- If there is no requested `id_expr` in the dictionary then:

- `'dictGet[Type]'` returns the content of the `'<null_value>'` element specified for the attribute in the dictionary configuration.
- `'dictGet[Type]OrDefault'` returns the value passed as the `'default_value_expr'` parameter.

ClickHouse throws an exception if it cannot parse the value of the attribute or the value does not match the attribute data type.

## Functions for Working with Yandex.Metrica Dictionaries

In order for the functions below to work, the server config must specify the paths and addresses for getting all the Yandex.Metrica dictionaries. The dictionaries are loaded at the first call of any of these functions. If the reference lists can't be loaded, an exception is thrown.

For information about creating reference lists, see the section “[Dictionaries](#)”.

## Multiple Geobases

ClickHouse supports working with multiple alternative geobases (regional hierarchies) simultaneously, in order to support various perspectives on which countries certain regions belong to.

The ‘clickhouse-server’ config specifies the file with the regional hierarchy::`<path_to_regions_hierarchy_file>/opt/geo/regions_hierarchy.txt</path_to_regions_hierarchy_file>`

Besides this file, it also searches for files nearby that have the `_` symbol and any suffix appended to the name (before the file extension).

For example, it will also find the file `/opt/geo/regions_hierarchy_ua.txt`, if present.

`ua` is called the dictionary key. For a dictionary without a suffix, the key is an empty string.

All the dictionaries are re-loaded in runtime (once every certain number of seconds, as defined in the `builtin_dictionaries_reload_interval` config parameter, or once an hour by default). However, the list of available dictionaries is defined one time, when the server starts.

All functions for working with regions have an optional argument at the end – the dictionary key. It is referred to as the geobase.

Example:

```
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
```

## regionToCity(id[, geobase])

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.

## regionToArea(id[, geobase])

Converts a region to an area (type 5 in the geobase). In every other way, this function is the same as 'regionToCity'.

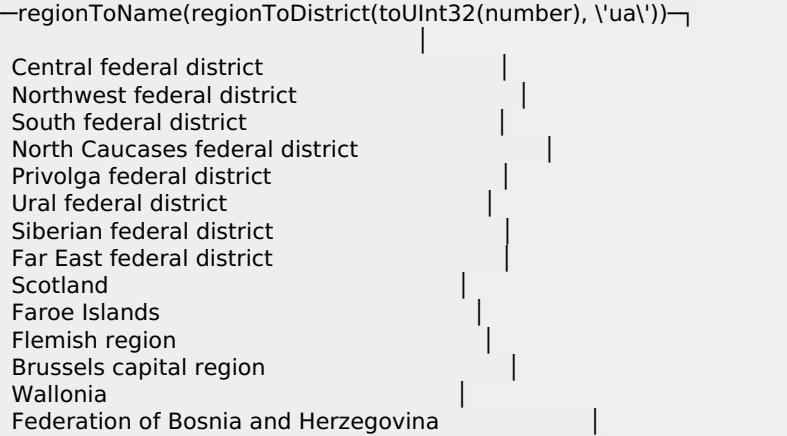
```
SELECT DISTINCT regionToName(regionToArea(toUInt32(number), 'ua'))  
FROM system.numbers  
LIMIT 15
```

```
regionToName(regionToArea(toUInt32(number), '\ua'))  
|  
Moscow and Moscow region  
St. Petersburg and Leningrad region  
Belgorod region  
Ivanovsk region  
Kaluga region  
Kostroma region  
Kursk region  
Lipetsk region  
Orlov region  
Ryazan region  
Smolensk region  
Tambov region  
Tver region  
Tula region
```

## regionToDistrict(id[, geobase])

Converts a region to a federal district (type 4 in the geobase). In every other way, this function is the same as 'regionToCity'.

```
SELECT DISTINCT regionToName(regionToDistrict(toUInt32(number), 'ua'))  
FROM system.numbers  
LIMIT 15
```



## regionToCountry(id[, geobase])

Converts a region to a country. In every other way, this function is the same as ‘regionToCity’.

Example: `regionToCountry(toUInt32(213)) = 225` converts Moscow (213) to Russia (225).

## regionToContinent(id[, geobase])

Converts a region to a continent. In every other way, this function is the same as ‘regionToCity’.

Example: `regionToContinent(toUInt32(213)) = 10001` converts Moscow (213) to Eurasia (10001).

## regionToTopContinent (#regiontotopcontinent)

Finds the highest continent in the hierarchy for the region.

### Syntax

```
regionToTopContinent(id[, geobase])
```

### Arguments

- `id` — Region ID from the Yandex geobase. [UInt32](#).
- `geobase` — Dictionary key. See [Multiple Geobases](#). [String](#). Optional.

### Returned value

- Identifier of the top level continent (the latter when you climb the hierarchy of regions).
- 0, if there is none.

Type: [UInt32](#).

## regionToPopulation(id[, geobase])

Gets the population for a region.

The population can be recorded in files with the geobase. See the section “External dictionaries”.

If the population is not recorded for the region, it returns 0.

In the Yandex geobase, the population might be recorded for child regions, but not for parent regions.

## regionIn(lhs, rhs[, geobase])

Checks whether a ‘lhs’ region belongs to a ‘rhs’ region. Returns a [UInt8](#) number equal to 1 if it belongs, or 0 if it does not belong.

The relationship is reflexive – any region also belongs to itself.

## regionHierarchy(id[, geobase])

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.

Example: `regionHierarchy(toUInt32(213)) = [213,1,3,225,10001,10000]`.

## regionToName(id[, lang])

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 does not exist, an empty string is returned.

`ua` and `uk` both mean Ukrainian.

## Functions for Implementing the IN Operator in, notIn, globalIn, globalNotIn

See the section [IN operators](#).

## arrayJoin function

This is a very unusual function.

Normal functions do not change a set of rows, but just change the values in each row (map).

Aggregate functions compress a set of rows (fold or reduce).

The ‘arrayJoin’ function takes each row and generates a set of rows (unfold).

This function takes an array as an argument, and propagates the source row to multiple rows for the number of elements in the array.

All the values in columns are simply copied, except the values in the column where this function is applied; it is replaced with the corresponding array value.

A query can use multiple `arrayJoin` functions. In this case, the transformation is performed multiple times.

Note the ARRAY JOIN syntax in the SELECT query, which provides broader possibilities.

Example:

```
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]

## Functions for Working with Geographical Coordinates

### greatCircleDistance

Calculates the distance between two points on the Earth's surface using the great-circle formula.

```
greatCircleDistance(lon1Deg, lat1Deg, lon2Deg, lat2Deg)
```

## Input parameters

- `lon1Deg` — Longitude of the first point in degrees. Range: [-180°, 180°].
- `lat1Deg` — Latitude of the first point in degrees. Range: [-90°, 90°].
- `lon2Deg` — Longitude of the second point in degrees. Range: [-180°, 180°].
- `lat2Deg` — Latitude of the second point in degrees. Range: [-90°, 90°].

Positive values correspond to North latitude and East longitude, and negative values correspond to South latitude and West longitude.

## Returned value

The distance between two points on the Earth's surface, in meters.

Generates an exception when the input parameter values fall outside of the range.

## Example

```
SELECT greatCircleDistance(55.755831, 37.617673, -55.755831, -37.617673)
```

```
greatCircleDistance(55.755831, 37.617673, -55.755831, -37.617673)←  
14132374.194975413 |
```

# geoDistance

Similar to `greatCircleDistance` but calculates the distance on WGS-84 ellipsoid instead of sphere. This is more precise approximation of the Earth Geoid.

The performance is the same as for `greatCircleDistance` (no performance drawback). It is recommended to use `geoDistance` to calculate the distances on Earth.

Technical note: for close enough points we calculate the distance using planar approximation with the metric on the tangent plane at the midpoint of the coordinates.

# greatCircleAngle

Calculates the central angle between two points on the Earth's surface using the great-circle formula.

```
greatCircleAngle(lon1Deg, lat1Deg, lon2Deg, lat2Deg)
```

## Input parameters

- `lon1Deg` — Longitude of the first point in degrees.
- `lat1Deg` — Latitude of the first point in degrees.
- `lon2Deg` — Longitude of the second point in degrees.
- `lat2Deg` — Latitude of the second point in degrees.

## Returned value

The central angle between two points in degrees.

## Example

```
SELECT greatCircleAngle(0, 0, 45, 0) AS arc
```

```
arc  
45 |
```

## pointInEllipses

Checks whether the point belongs to at least one of the ellipses.

Coordinates are geometric in the Cartesian coordinate system.

```
pointInEllipses(x, y, x0, y0, a0, b0,...,xn, yn, an, bn)
```

### Input parameters

- $x, y$  — Coordinates of a point on the plane.
- $x_i, y_i$  — Coordinates of the center of the  $i$ -th ellipsis.
- $a_i, b_i$  — Axes of the  $i$ -th ellipsis in units of  $x, y$  coordinates.

The input parameters must be  $2+4\cdot n$ , where  $n$  is the number of ellipses.

### Returned values

1 if the point is inside at least one of the ellipses; 0 if it is not.

## Example

```
SELECT pointInEllipses(10., 10., 10., 9.1, 1., 0.9999)
```

```
pointInEllipses(10., 10., 10., 9.1, 1., 0.9999)  
1 |
```

## pointInPolygon

Checks whether the point belongs to the polygon on the plane.

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

### Input values

- $(x, y)$  — Coordinates of a point on the plane. Data type — **Tuple** — A tuple of two numbers.
- $[(a, b), (c, d) ...]$  — Polygon vertices. Data type — **Array**. Each vertex is represented by a pair of coordinates  $(a, b)$ . Vertices should be specified in a clockwise or counterclockwise order. The minimum number of vertices is 3. The polygon must be constant.
- The function also supports polygons with holes (cut out sections). In this case, add polygons that define the cut out sections using additional arguments of the function. The function does not support non-simply-connected polygons.

## Returned values

1 if the point is inside the polygon, 0 if it is not.

If the point is on the polygon boundary, the function may return either 0 or 1.

## Example

```
SELECT pointInPolygon((3., 3.), [(6, 0), (8, 4), (5, 8), (0, 2)]) AS res
```

```
res  
1 |
```

# Functions for Working with Geohash

**Geohash** is the geocode system, which subdivides Earth's surface into buckets of grid shape and encodes each cell into a short string of letters and digits. It is a hierarchical data structure, so the longer is the geohash string, the more precise is the geographic location.

If you need to manually convert geographic coordinates to geohash strings, you can use [geohash.org](#).

## geohashEncode

Encodes latitude and longitude as a **geohash**-string.

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

## Input values

- longitude - longitude part of the coordinate you want to encode. Floating in range [-180°, 180°]
- latitude - latitude part of the coordinate you want to encode. Floating in range [-90°, 90°]
- precision - Optional, length of the resulting encoded string, defaults to 12. Integer in range [1, 12]. Any value less than 1 or greater than 12 is silently converted to 12.

## Returned values

- alphanumeric **String** of encoded coordinate (modified version of the base32-encoding alphabet is used).

## Example

```
SELECT geohashEncode(-5.60302734375, 42.593994140625, 0) AS res;
```

```
res  
ezs42d000000 |
```

## geohashDecode

Decodes any **geohash**-encoded string into longitude and latitude.

## Input values

- encoded string - geohash-encoded string.

## Returned values

- (longitude, latitude) - 2-tuple of `Float64` values of longitude and latitude.

## Example

```
SELECT geohashDecode('ezs42') AS res;
```

```
res
(-5.60302734375,42.60498046875) |
```

## geohashesInBox

Returns an array of `geohash`-encoded strings of given precision that fall inside and intersect boundaries of given box, basically a 2D grid flattened into array.

## Syntax

```
geohashesInBox(longitude_min, latitude_min, longitude_max, latitude_max, precision)
```

## Arguments

- `longitude_min` — Minimum longitude. Range: [-180°, 180°]. Type: `Float`.
- `latitude_min` — Minimum latitude. Range: [-90°, 90°]. Type: `Float`.
- `longitude_max` — Maximum longitude. Range: [-180°, 180°]. Type: `Float`.
- `latitude_max` — Maximum latitude. Range: [-90°, 90°]. Type: `Float`.
- `precision` — Geohash precision. Range: [1, 12]. Type: `UInt8`.

## Note

All coordinate parameters must be of the same type: either `Float32` or `Float64`.

## Returned values

- Array of precision-long strings of geohash-boxes covering provided area, you should not rely on order of items.
- [] - Empty array if minimum latitude and longitude values aren't less than corresponding maximum values.

Type: `Array(String)`.

## Note

Function throws an exception if resulting array is over 10'000'000 items long.

## Example

Query:

```
SELECT geohashesInBox(24.48, 40.56, 24.785, 40.81, 4) AS thasos;
```

Result:

```
thasos  
['sx1q','sx1r','sx32','sx1w','sx1x','sx38'] |
```

## Functions for Working with H3 Indexes

**H3** is a geographical indexing system where Earth's surface divided into a grid of even hexagonal cells. This system is hierarchical, i. e. each hexagon on the top level ("parent") can be splitted into seven even but smaller ones ("children"), and so on.

The level of the hierarchy is called **resolution** and can receive a value from **0** till **15**, where **0** is the **base level** with the largest and coarsest cells.

A latitude and longitude pair can be transformed to a 64-bit H3 index, identifying a grid cell.

The H3 index is used primarily for bucketing locations and other geospatial manipulations.

The full description of the H3 system is available at [the Uber Engeneering site](#).

### h3IsValid

Verifies whether the number is a valid **H3** index.

#### Syntax

```
h3IsValid(h3index)
```

#### Parameter

- **h3index** — Hexagon index number. Type: **UInt64**.

#### Returned values

- **1** — The number is a valid H3 index.
- **0** — The number is not a valid H3 index.

Type: **UInt8**.

#### Example

Query:

```
SELECT h3IsValid(630814730351855103) AS h3isValid;
```

Result:

```
h3isValid  
1 |
```

# **h3GetResolution**

Defines the resolution of the given **H3** index.

## **Syntax**

```
h3GetResolution(h3index)
```

## **Parameter**

- `h3index` — Hexagon index number. Type: **UInt64**.

## **Returned values**

- Index resolution. Range: [0, 15].
- If the index is not valid, the function returns a random value. Use **h3IsValid** to verify the index.

Type: **UInt8**.

## **Example**

Query:

```
SELECT h3GetResolution(639821929606596015) AS resolution;
```

Result:

resolution
14

# **h3EdgeAngle**

Calculates the average length of the **H3** hexagon edge in grades.

## **Syntax**

```
h3EdgeAngle(resolution)
```

## **Parameter**

- `resolution` — Index resolution. Type: **UInt8**. Range: [0, 15].

## **Returned values**

- The average length of the **H3** hexagon edge in grades. Type: **Float64**.

## **Example**

Query:

```
SELECT h3EdgeAngle(10) AS edgeAngle;
```

Result:

```
h3EdgeAngle(10)  
0.0005927224846720883 |
```

## h3EdgeLengthM

Calculates the average length of the **H3** hexagon edge in meters.

### Syntax

```
h3EdgeLengthM(resolution)
```

### Parameter

- `resolution` — Index resolution. Type: **UInt8**. Range: [0, 15].

### Returned values

- The average length of the **H3** hexagon edge in meters. Type: **Float64**.

### Example

Query:

```
SELECT h3EdgeLengthM(15) AS edgeLengthM;
```

Result:

```
edgeLengthM  
0.509713273 |
```

## geoToH3

Returns **H3** point index (`lon`, `lat`) with specified resolution.

### Syntax

```
geoToH3(lon, lat, resolution)
```

### Arguments

- `lon` — Longitude. Type: **Float64**.
- `lat` — Latitude. Type: **Float64**.
- `resolution` — Index resolution. Range: [0, 15]. Type: **UInt8**.

### Returned values

- Hexagon index number.
- 0 in case of error.

Type: **UInt64**.

### Example

Query:

```
SELECT geoToH3(37.79506683, 55.71290588, 15) AS h3Index;
```

Result:

```
h3Index  
644325524701193974
```

## h3ToGeo

Returns the geographical coordinates of longitude and latitude corresponding to the provided H3 index.

### Syntax

```
h3ToGeo(h3Index)
```

### Arguments

- h3Index — H3 Index. **UInt64**.

### Returned values

- A tuple consisting of two values: tuple(lon,lat). lon — Longitude. **Float64**. lat — Latitude. **Float64**.

### Example

Query:

```
SELECT h3ToGeo(644325524701193974) AS coordinates;
```

Result:

```
coordinates  
(37.79506616830252,55.71290243145668)
```

## h3ToGeoBoundary

Returns array of pairs (lon, lat), which corresponds to the boundary of the provided H3 index.

### Syntax

```
h3ToGeoBoundary(h3Index)
```

### Arguments

- h3Index — H3 Index. Type: **UInt64**.

### Returned values

- Array of pairs '(lon, lat)'.  
Type: **Array(Float64, Float64)**.

### Example

Query:

```
SELECT h3ToGeoBoundary(644325524701193974) AS coordinates;
```

Result:

```
└─h3ToGeoBoundary(599686042433355775)─  
  └─[(37.2713558667319,-121.91508032705622),(37.353926450852256,-121.8622232890249),(37.42834118609435,-121.92354999630156),(37.42012867767779,-122.03773496427027),(37.33755608435299,-122.090428929044),(37.26319797461824,-122.02910130919001)] └
```

## h3kRing

Lists all the **H3** hexagons in the radius of **k** from the given hexagon in random order.

### Syntax

```
h3kRing(h3index, k)
```

### Arguments

- **h3index** — Hexagon index number. Type: **UInt64**.
- **k** — Radius. Type: **integer**

### Returned values

- Array of H3 indexes.

Type: **Array(UInt64)**.

### Example

Query:

```
SELECT arrayJoin(h3kRing(644325529233966508, 1)) AS h3index;
```

Result:

```
└─h3index─  
  644325529233966508  
  644325529233966497  
  644325529233966510  
  644325529233966504  
  644325529233966509  
  644325529233966355  
  644325529233966354
```

## h3GetBaseCell

Returns the base cell number of the **H3** index.

### Syntax

```
h3GetBaseCell(index)
```

## Parameter

- index — Hexagon index number. Type: **UInt64**.

## Returned value

- Hexagon base cell number.

Type: **UInt8**.

## Example

Query:

```
SELECT h3GetBaseCell(612916788725809151) AS basecell;
```

Result:

```
basecell
12 |
```

## h3HexAreaM2

Returns average hexagon area in square meters at the given resolution.

## Syntax

```
h3HexAreaM2(resolution)
```

## Parameter

- resolution — Index resolution. Range: [0, 15]. Type: **UInt8**.

## Returned value

- Area in square meters.

Type: **Float64**.

## Example

Query:

```
SELECT h3HexAreaM2(13) AS area;
```

Result:

```
area
43.9 |
```

## h3IndexesAreNeighbors

Returns whether or not the provided **H3** indexes are neighbors.

## Syntax

```
h3IndexesAreNeighbors(index1, index2)
```

## Arguments

- `index1` — Hexagon index number. Type: `UInt64`.
- `index2` — Hexagon index number. Type: `UInt64`.

## Returned value

- `1` — Indexes are neighbours.
- `0` — Indexes are not neighbours.

Type: `UInt8`.

## Example

Query:

```
SELECT h3IndexesAreNeighbors(617420388351344639, 617420388352655359) AS n;
```

Result:

```
+-----+  
| 1 |  
+-----+
```

## h3ToChildren

Returns an array of child indexes for the given `H3` index.

## Syntax

```
h3ToChildren(index, resolution)
```

## Arguments

- `index` — Hexagon index number. Type: `UInt64`.
- `resolution` — Index resolution. Range: [0, 15]. Type: `UInt8`.

## Returned values

- Array of the child H3-indexes.

Type: `Array(UInt64)`.

## Example

Query:

```
SELECT h3ToChildren(599405990164561919, 6) AS children;
```

Result:

```
└─children
```

```
[603909588852408319, 603909588986626047, 603909589120843775, 603909589255061503, 603909589389279231,  
603909589523496959, 603909589657714687] |
```

## h3ToParent

Returns the parent (coarser) index containing the given H3 index.

### Syntax

```
h3ToParent(index, resolution)
```

### Arguments

- `index` — Hexagon index number. Type: [UInt64](#).
- `resolution` — Index resolution. Range: [0, 15]. Type: [UInt8](#).

### Returned value

- Parent H3 index.

Type: [UInt64](#).

### Example

Query:

```
SELECT h3ToParent(599405990164561919, 3) AS parent;
```

Result:

```
parent  
590398848891879423
```

## h3ToString

Converts the `H3Index` representation of the index to the string representation.

```
h3ToString(index)
```

### Parameter

- `index` — Hexagon index number. Type: [UInt64](#).

### Returned value

- String representation of the H3 index.

Type: [String](#).

### Example

Query:

```
SELECT h3ToString(617420388352917503) AS h3_string;
```

Result:

```
h3_string  
89184926cdbffff |
```

## stringToH3

Converts the string representation to the `H3Index` (`UInt64`) representation.

### Syntax

```
stringToH3(index_str)
```

### Parameter

- `index_str` — String representation of the H3 index. Type: `String`.

### Returned value

- Hexagon index number. Returns 0 on error. Type: `UInt64`.

### Example

Query:

```
SELECT stringToH3('89184926cc3ffff') AS index;
```

Result:

```
index  
617420388351344639 |
```

## h3GetResolution

Returns the resolution of the `H3` index.

### Syntax

```
h3GetResolution(index)
```

### Parameter

- `index` — Hexagon index number. Type: `UInt64`.

### Returned value

- Index resolution. Range: [0, 15]. Type: `UInt8`.

### Example

Query:

```
SELECT h3GetResolution(617420388352917503) AS res;
```

Result:

```
res  
9 |
```

## h3IsResClassIII

Returns whether **H3** index has a resolution with Class III orientation.

### Syntax

```
h3IsResClassIII(index)
```

### Parameter

- `index` — Hexagon index number. Type: [UInt64](#).

### Returned value

- `1` — Index has a resolution with Class III orientation.
- `0` — Index doesn't have a resolution with Class III orientation.

Type: [UInt8](#).

### Example

Query:

```
SELECT h3IsResClassIII(617420388352917503) AS res;
```

Result:

```
res  
1 |
```

## h3IsPentagon

Returns whether this **H3** index represents a pentagonal cell.

### Syntax

```
h3IsPentagon(index)
```

### Parameter

- `index` — Hexagon index number. Type: [UInt64](#).

### Returned value

- `1` — Index represents a pentagonal cell.
- `0` — Index doesn't represent a pentagonal cell.

Type: [UInt8](#).

## Example

Query:

```
SELECT h3IsPentagon(644721767722457330) AS pentagon;
```

Result:

```
pentagon
0 |
```

## h3GetFaces

Returns icosahedron faces intersected by a given [H3](#) index.

## Syntax

```
h3GetFaces(index)
```

## Parameter

- index — Hexagon index number. Type: [UInt64](#).

## Returned values

- Array containing icosahedron faces intersected by a given H3 index.

Type: [Array\(UInt64\)](#).

## Example

Query:

```
SELECT h3GetFaces(599686042433355775) AS faces;
```

Result:

```
faces
[7] |
```

toc\_title: S2 Geometry

## Functions for Working with Nullable Values

### isNull

Checks whether the argument is [NULL](#).

```
isNull(x)
```

Alias: `ISNULL`.

## Arguments

- `x` — A value with a non-compound data type.

## Returned value

- `1` if `x` is `NULL`.
- `0` if `x` is not `NULL`.

## Example

Input table

x	y
1	NULL
2	3

Query

```
SELECT x FROM t_null WHERE isNull(y);
```

x
1

## isNotNull

Checks whether the argument is `NULL`.

```
isNotNull(x)
```

## Arguments:

- `x` — A value with a non-compound data type.

## Returned value

- `0` if `x` is `NULL`.
- `1` if `x` is not `NULL`.

## Example

Input table

x	y
1	NULL
2	3

Query

```
SELECT x FROM t_null WHERE isNotNull(y);
```

x
2

## coalesce

Checks from left to right whether `NULL` arguments were passed and returns the first non-`NULL` argument.

```
coalesce(x,...)
```

### Arguments:

- Any number of parameters of a non-compound type. All parameters must be compatible by data type.

### Returned values

- The first non-`NULL` argument.
- `NULL`, if all arguments are `NULL`.

### Example

Consider a list of contacts that may specify multiple ways to contact a customer.

name	mail	phone	icq
client 1	NULL	123-45-67	123
client 2	NULL	NULL	NULL

The mail and phone fields are of type String, but the icq field is UInt32, so it needs to be converted to String.

Get the first available contact method for the customer from the contact list:

```
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

## ifNull

Returns an alternative value if the main argument is `NULL`.

```
ifNull(x,alt)
```

### Arguments:

- `x` — The value to check for `NULL`.
- `alt` — The value that the function returns if `x` is `NULL`.

### Returned values

- The value `x`, if `x` is not `NULL`.
- The value `alt`, if `x` is `NULL`.

## Example

```
SELECT ifNull('a', 'b');
```

ifNull('a', 'b')  
  |  
  a

```
SELECT ifNull(NULL, 'b');
```

ifNull(NULL, 'b')  
  |  
  b

## nullIf

Returns `NULL` if the arguments are equal.

```
nullIf(x, y)
```

### Arguments:

`x, y` — Values for comparison. They must be compatible types, or ClickHouse will generate an exception.

### Returned values

- `NULL`, if the arguments are equal.
- The `x` value, if the arguments are not equal.

## Example

```
SELECT nullIf(1, 1);
```

nullIf(1, 1)  
  |  
  NULL

```
SELECT nullIf(1, 2);
```

nullIf(1, 2)  
  |  
  1

## assumeNotNull

Results in an equivalent non-`Nullable` value for a `Nullable` type. In case the original value is `NULL` the result is undetermined. See also `ifNull` and `coalesce` functions.

```
assumeNotNull(x)
```

## Arguments:

- `x` — The original value.

## Returned values

- The original value from the non-`Nullable` type, if it is not `NULL`.
- Implementation specific result if the original value was `NULL`.

## Example

Consider the `t_null` table.

```
SHOW CREATE TABLE t_null;
```

```
CREATE TABLE default.t_null ( x Int8, y Nullable(Int8) ENGINE = TinyLog |
```

X	Y
1	NULL
2	3

Apply the `assumeNotNull` function to the `y` column.

```
SELECT assumeNotNull(y) FROM t_null;
```

```
assumeNotNull(y)|  
0 |  
3 |
```

```
SELECT toTypeName(assumeNotNull(y)) FROM t_null;
```

```
toTypeName(assumeNotNull(y))|  
Int8 |  
Int8 |
```

## toNullable

Converts the argument type to `Nullable`.

```
toNullable(x)
```

## Arguments:

- `x` — The value of any non-compound type.

## Returned value

- The input value with a `Nullable` type.

## Example

```
SELECT toTypeName(10);
```

```
toTypeName(10)  
|  
UInt8
```

```
SELECT toTypeName(toNullable(10));
```

```
toTypeName(toNullable(10))  
|  
Nullable(UInt8)
```

## Machine Learning Functions

### evalMLMethod

Prediction using fitted regression models uses `evalMLMethod` function. See link in [linearRegression](#).

### stochasticLinearRegression

The `stochasticLinearRegression` aggregate function implements stochastic gradient descent method using linear model and MSE loss function. Uses `evalMLMethod` to predict on new data.

### stochasticLogisticRegression

The `stochasticLogisticRegression` aggregate function implements stochastic gradient descent method for binary classification problem. Uses `evalMLMethod` to predict on new data.

## Introspection Functions

You can use functions described in this chapter to introspect [ELF](#) and [DWARF](#) for query profiling.

### Warning

These functions are slow and may impose security considerations.

For proper operation of introspection functions:

- Install the `clickhouse-common-static-dbg` package.
- Set the `allow_introspection_functions` setting to 1.

For security reasons introspection functions are disabled by default.

ClickHouse saves profiler reports to the `trace_log` system table. Make sure the table and profiler are configured properly.

# addressToLine

Converts virtual memory address inside ClickHouse server process to the filename and the line number in ClickHouse source code.

If you use official ClickHouse packages, you need to install the `clickhouse-common-static-dbg` package.

## Syntax

```
addressToLine(address_of_binary_instruction)
```

## Arguments

- `address_of_binary_instruction` (`UInt64`) — Address of instruction in a running process.

## Returned value

- Source code filename and the line number in this file delimited by colon.

For example, `/build/obj-x86_64-linux-gnu/../src/Common/ThreadPool.cpp:199`, where `199` is a line number.

- Name of a binary, if the function couldn't find the debug information.
- Empty string, if the address is not valid.

Type: `String`.

## Example

Enabling introspection functions:

```
SET allow_introspection_functions=1;
```

Selecting the first string from the `trace_log` system table:

```
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:	[140658411141617,94784174532828,94784076370703,94784076372094,94784076361020,94784175007680,14065841116251,140658403895439]

The `trace` field contains the stack trace at the moment of sampling.

Getting the source code filename and the line number for a single address:

```
SELECT addressToLine(94784076370703) \G;
```

Row 1:

```
addressToLine(94784076370703): /build/obj-x86_64-linux-gnu/../src/Common/ThreadPool.cpp:199
```

Applying the function to the whole stack trace:

```
SELECT
    arrayStringConcat(arrayMap(x -> addressToLine(x), trace), '\n') AS trace_source_code_lines
FROM system.trace_log
LIMIT 1
\G
```

The `arrayMap` function allows to process each individual element of the `trace` array by the `addressToLine` function. The result of this processing you see in the `trace_source_code_lines` column of output.

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

Converts virtual memory address inside ClickHouse server process to the symbol from ClickHouse object files.

### Syntax

```
addressToSymbol(address_of_binary_instruction)
```

### Arguments

- `address_of_binary_instruction` (`UInt64`) — Address of instruction in a running process.

### Returned value

- Symbol from ClickHouse object files.
- Empty string, if the address is not valid.

Type: `String`.

### Example

Enabling introspection functions:

```
SET allow_introspection_functions=1;
```

Selecting the first string from the `trace_log` system table:

```
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, 9413881501224, 9413881502091, 94138814222988, 94138806823642, 94138814457211, 94138806823642, 94138814457211, 94138806823642, 94138806795179, 94138806796, 94138753770094, 94138753771646, 94138753760572, 94138852407232, 140399185266395, 140399178045583]
```

The trace field contains the stack trace at the moment of sampling.

Getting a symbol for a single address:

```
SELECT addressToSymbol(94138803686098) \G;
```

Row 1:

```
addressToSymbol(94138803686098):
_ZNK2DB24IAggregateFunctionHelperINS_20AggregateFunctionSumImmNS_24AggregateFunctionSumDataImEEEEEE1
9addBatchSinglePlaceEmPcPPKNS_7IColumnEPNS_5ArenaE
```

Applying the function to the whole stack trace:

```
SELECT
  arrayStringConcat(arrayMap(x -> addressToSymbol(x), trace), '\n') AS trace_symbols
FROM system.trace_log
LIMIT 1
\G
```

The `arrayMap` function allows to process each individual element of the trace array by the `addressToSymbols` function. The result of this processing you see in the `trace_symbols` column of output.

Row 1:

```
trace_symbols:
_ZNK2DB24IAggregateFunctionHelperINS_20AggregateFunctionSumImmNS_24AggregateFunctionSumDataImEEEEEE1
9addBatchSinglePlaceEmPcPPKNS_7IColumnEPNS_5ArenaE
_ZNK2DB10Aggregator21executeWithoutKeyImplERPcmPNS0_28AggregateFunctionInstructionEPNS_5ArenaE
_ZN2DB10Aggregator14executeOnBlockESt6vectorIN3COWINS_7IColumnEE13immutable_ptrIS3_EESaIS6_EEmRNS_2
2AggregatedDataVariantsERS1_IPKS3_SaISC_EERS1_ISE_SaISE_EERb
_ZN2DB10Aggregator14executeOnBlockERKNS_5BlockERNS_22AggregatedDataVariantsERSt6vectorIPKNS_7IColumn
ESaIS9_EERS6_ISB_SaISB_EERb
_ZN2DB10Aggregator7executeERKSt10shared_ptrINS_17IBlockInputStreamEERNS_22AggregatedDataVariantsE
_ZN2DB27AggregatingBlockInputStream8readImplEv
_ZN2DB17IBlockInputStream4readEv
_ZN2DB26ExpressionBlockInputStream8readImplEv
_ZN2DB17IBlockInputStream4readEv
_ZN2DB26ExpressionBlockInputStream8readImplEv
_ZN2DB17IBlockInputStream4readEv
_ZN2DB28AsynchronousBlockInputStream9calculateEv
_ZNSt17_Function_handlerIFvvEZN2DB28AsynchronousBlockInputStream4nextEvEUIvE_E9_M_invokeERKSt9_Any_dat
a
_ZN14ThreadPoolImplI20ThreadFromGlobalPoolE6workerESt14_List_iteratorIS0_E
_ZZN20ThreadFromGlobalPoolC4IZN14ThreadPoolImplIS_E12scheduleImplIvEET_St8functionIFvvEEiSt8optionalImEEU
vE1_JEEEOS4_DpOT0_ENKUIvE_cIEv
_ZN14ThreadPoolImplISt6threadE6workerESt14_List_iteratorIS0_E
execute_native_thread_routine
start_thread
clone
```

# demangle

Converts a symbol that you can get using the [addressToSymbol](#) function to the C++ function name.

## Syntax

```
demangle(symbol)
```

## Arguments

- `symbol` ([String](#)) — Symbol from an object file.

## Returned value

- Name of the C++ function.
- Empty string if a symbol is not valid.

Type: [String](#).

## Example

Enabling introspection functions:

```
SET allow_introspection_functions=1;
```

Selecting the first string from the `trace_log` system table:

```
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]
```

The `trace` field contains the stack trace at the moment of sampling.

Getting a function name for a single address:

```
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
```

Applying the function to the whole stack trace:

```

SELECT
    arrayStringConcat(arrayMap(x -> demangle(addressToSymbol(x)), trace), '\n') AS trace_functions
FROM system.trace_log
LIMIT 1
\G

```

The `arrayMap` function allows to process each individual element of the `trace` array by the `demangle` function. The result of this processing you see in the `trace_functions` column of output.

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<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<std::thread>)
execute_native_thread_routine
start_thread
clone

```

## tid

Returns id of the thread, in which current `Block` is processed.

### Syntax

```
tid()
```

### Returned value

- Current thread id. `UInt64`.

### Example

Query:

```
SELECT tid();
```

Result:

```
tid()—  
3878 |
```

## logTrace

Emits trace log message to server log for each [Block](#).

### Syntax

```
logTrace('message')
```

### Arguments

- **message** — Message that is emitted to server log. [String](#).

### Returned value

- Always returns 0.

### Example

Query:

```
SELECT logTrace('logTrace message');
```

Result:

```
logTrace('logTrace message')—  
0 |
```

## Functions for Working with Tuples

### tuple

A function that allows grouping multiple columns.

For columns with the types T1, T2, ..., it returns a Tuple(T1, T2, ...) type tuple containing these columns.

There is no cost to execute the function.

Tuples are normally used as intermediate values for an argument of IN operators, or for creating a list of formal parameters of lambda functions. Tuples can't be written to a table.

The function implements the operator (x, y, ...).

### Syntax

```
tuple(x, y, ...)
```

### tupleElement

A function that allows getting a column from a tuple.

'N' is the column index, starting from 1. N must be a constant. 'N' must be a constant. 'N' must be a strict positive integer no greater than the size of the tuple.

There is no cost to execute the function.

The function implements the operator `x.N`.

## Syntax

```
tupleElement(tuple, n)
```

# untuple

Performs syntactic substitution of `tuple` elements in the call location.

## Syntax

```
untuple(x)
```

You can use the `EXCEPT` expression to skip columns as a result of the query.

## Arguments

- `x` — A `tuple` function, column, or tuple of elements. `Tuple`.

## Returned value

- None.

## Examples

Input table:

key	v1	v2	v3	v4	v5	v6
1	10	20	40	30	15	(33,'ab')
2	25	65	70	40	6	(44,'cd')
3	57	30	20	10	5	(55,'ef')
4	55	12	7	80	90	(66,'gh')
5	30	50	70	25	55	(77,'kl')

Example of using a Tuple-type column as the `untuple` function parameter:

Query:

```
SELECT untuple(v6) FROM kv;
```

Result:

_ut_1	_ut_2
33	ab
44	cd
55	ef
66	gh
77	kl

Note: the names are implementation specific and are subject to change. You should not assume specific names of the columns after application of the `untuple`.

Example of using an `EXCEPT` expression:

Query:

```
SELECT untuple(* EXCEPT (v2, v3),) FROM kv;
```

Result:

key	v1	v4	v5	v6
1	10	30	15	(33,'ab')
2	25	40	6	(44,'cd')
3	57	10	5	(55,'ef')
4	55	80	90	(66,'gh')
5	30	25	55	(77,'kl')

## See Also

- [Tuple](#)

## tupleHammingDistance

Returns the [Hamming Distance](#) between two tuples of the same size.

### Syntax

```
tupleHammingDistance(tuple1, tuple2)
```

### Arguments

- `tuple1` — First tuple. [Tuple](#).
- `tuple2` — Second tuple. [Tuple](#).

Tuples should have the same type of the elements.

### Returned value

- The Hamming distance.

Type: [UInt8](#).

### Examples

Query:

```
SELECT tupleHammingDistance((1, 2, 3), (3, 2, 1)) AS HammingDistance;
```

Result:

HammingDistance
2

Can be used with [MinHash](#) functions for detection of semi-duplicate strings:

```
SELECT tupleHammingDistance(wordShingleMinHash(string), wordShingleMinHashCaseInsensitive(string)) as HammingDistance FROM (SELECT 'ClickHouse is a column-oriented database management system for online analytical processing of queries.' AS string);
```

Result:

```
└─HammingDistance─  
  2 |
```

## tupleToNameValuePairs

Turns a named tuple into an array of (name, value) pairs. For a Tuple(a T, b T, ..., c T) returns Array(Tuple(String, T), ...)

in which the Strings represents the named fields of the tuple and T are the values associated with those names. All values in the tuple should be of the same type.

### Syntax

```
tupleToNameValuePairs(tuple)
```

\*\*Arguments\*\*

- `tuple` — Named tuple. [Tuple](#sql-reference-data-types-tuple-md) with any types of values.

\*\*Returned value\*\*

- An array with (name, value) pairs.

Type: [Array](#sql-reference-data-types-array-md)([Tuple](#sql-reference-data-types-tuple-md)([String](#sql-reference-data-types-string-md), ...)).

\*\*Example\*\*

Query:

```
```sql  
CREATE TABLE tupletest (`col` Tuple(user_ID UInt64, session_ID UInt64) ENGINE = Memory;  
INSERT INTO tupletest VALUES (tuple( 100, 2502)), (tuple(1,100));  
SELECT tupleToNameValuePairs(col) FROM tupletest;
```

Result:

```
└─tupleToNameValuePairs(col)─  
  [ ('user_ID',100), ('session_ID',2502) ] |  
  [ ('user_ID',1), ('session_ID',100) ] └
```

It is possible to transform columns to rows using this function:

```
CREATE TABLE tupletest (`col` Tuple(CPU Float64, Memory Float64, Disk Float64)) ENGINE = Memory;  
INSERT INTO tupletest VALUES(tuple(3.3, 5.5, 6.6));  
SELECT arrayJoin(tupleToNameValuePairs(col))FROM tupletest;
```

Result:

```
└─arrayJoin(tupleToNameValuePairs(col))─  
  ('CPU',3.3) |  
  ('Memory',5.5) |  
  ('Disk',6.6) └
```

If you pass a simple tuple to the function, ClickHouse uses the indexes of the values as their names:

```
SELECT tupleToNameValuePairs(tuple(3, 2, 1));
```

Result:

```
tupleToNameValuePairs(tuple(3, 2, 1))—  
[('1',3),('2',2),('3',1)] |
```

```
### tuplePlus {#tupleplus}
```

Calculates the sum of corresponding values of two tuples of the same size.

**\*\*Syntax\*\***

```
```sql  
tuplePlus(tuple1, tuple2)
```

Alias: vectorSum.

## Arguments

- `tuple1` — First tuple. [Tuple](#).
- `tuple2` — Second tuple. [Tuple](#).

## Returned value

- Tuple with the sum.

Type: [Tuple](#).

## Example

Query:

```
SELECT tuplePlus((1, 2), (2, 3));
```

Result:

```
tuplePlus((1, 2), (2, 3))—  
(3,5) |
```

# tupleMinus

Calculates the subtraction of corresponding values of two tuples of the same size.

## Syntax

```
tupleMinus(tuple1, tuple2)
```

Alias: vectorDifference.

## Arguments

- `tuple1` — First tuple. [Tuple](#).
- `tuple2` — Second tuple. [Tuple](#).

## Returned value

- Tuple with the result of subtraction.

Type: **Tuple**.

### Example

Query:

```
SELECT tupleMinus((1, 2), (2, 3));
```

Result:

```
tupleMinus((1, 2), (2, 3))  
(-1,-1)
```

## tupleMultiply

Calculates the multiplication of corresponding values of two tuples of the same size.

### Syntax

```
tupleMultiply(tuple1, tuple2)
```

### Arguments

- `tuple1` — First tuple. **Tuple**.
- `tuple2` — Second tuple. **Tuple**.

### Returned value

- Tuple with the multiplication.

Type: **Tuple**.

### Example

Query:

```
SELECT tupleMultiply((1, 2), (2, 3));
```

Result:

```
tupleMultiply((1, 2), (2, 3))  
(2,6)
```

## tupleDivide

Calculates the division of corresponding values of two tuples of the same size. Note that division by zero will return `inf`.

### Syntax

```
tupleDivide(tuple1, tuple2)
```

## Arguments

- `tuple1` — First tuple. [Tuple](#).
- `tuple2` — Second tuple. [Tuple](#).

## Returned value

- Tuple with the result of division.

Type: [Tuple](#).

## Example

Query:

```
SELECT tupleDivide((1, 2), (2, 3));
```

Result:

```
tupleDivide((1, 2), (2, 3))—  
| (0.5,0.6666666666666666)
```

# tupleNegate

Calculates the negation of the tuple values.

## Syntax

```
tupleNegate(tuple)
```

## Arguments

- `tuple` — [Tuple](#).

## Returned value

- Tuple with the result of negation.

Type: [Tuple](#).

## Example

Query:

```
SELECT tupleNegate((1, 2));
```

Result:

```
tupleNegate((1, 2))—  
| (-1,-2)
```

# tupleMultiplyByNumber

Returns a tuple with all values multiplied by a number.

## Syntax

```
tupleMultiplyByNumber(tuple, number)
```

## Arguments

- `tuple` — [Tuple](#).
- `number` — Multiplier. [Int/UInt](#), [Float](#) or [Decimal](#).

## Returned value

- Tuple with multiplied values.

Type: [Tuple](#).

## Example

Query:

```
SELECT tupleMultiplyByNumber((1, 2), -2.1);
```

Result:

```
tupleMultiplyByNumber((1, 2), -2.1)
(-2.1,-4.2)
```

## tupleDivideByNumber

Returns a tuple with all values divided by a number. Note that division by zero will return [inf](#).

## Syntax

```
tupleDivideByNumber(tuple, number)
```

## Arguments

- `tuple` — [Tuple](#).
- `number` — Divider. [Int/UInt](#), [Float](#) or [Decimal](#).

## Returned value

- Tuple with divided values.

Type: [Tuple](#).

## Example

Query:

```
SELECT tupleDivideByNumber((1, 2), 0.5);
```

Result:

```
tupleDivideByNumber((1, 2), 0.5)─  
     |  
     (2,4)
```

## dotProduct

Calculates the scalar product of two tuples of the same size.

### Syntax

```
dotProduct(tuple1, tuple2)
```

Alias: `scalarProduct`.

### Arguments

- `tuple1` — First tuple. [Tuple](#).
- `tuple2` — Second tuple. [Tuple](#).

### Returned value

- Scalar product.

Type: [Int/UInt](#), [Float](#) or [Decimal](#).

### Example

Query:

```
SELECT dotProduct((1, 2), (2, 3));
```

Result:

```
dotProduct((1, 2), (2, 3))─  
     |  
     8
```

## L1Norm

Calculates the sum of absolute values of a tuple.

### Syntax

```
L1Norm(tuple)
```

Alias: `normL1`.

### Arguments

- `tuple` — [Tuple](#).

### Returned value

- L1-norm or [taxicab geometry](#) distance.

Type: [UInt](#), [Float](#) or [Decimal](#).

## Example

Query:

```
SELECT L1Norm((1, 2));
```

Result:

```
L1Norm((1, 2))—  
3 |
```

## L2Norm

Calculates the square root of the sum of the squares of the tuple values.

### Syntax

```
L2Norm(tuple)
```

Alias: `normL2`.

### Arguments

- `tuple` — [Tuple](#).

### Returned value

- L2-norm or [Euclidean distance](#).

Type: [Float](#).

## Example

Query:

```
SELECT L2Norm((1, 2));
```

Result:

```
L2Norm((1, 2))—  
2.23606797749979 |
```

## LinfNorm

Calculates the maximum of absolute values of a tuple.

### Syntax

```
LinfNorm(tuple)
```

Alias: `normLinf`.

### Arguments

- `tuple` — [Tuple](#).

## Returned value

- Linf-norm or the maximum absolute value.

Type: **Float**.

## Example

Query:

```
SELECT LinfNorm((1, -2));
```

Result:

```
└─LinfNorm((1, -2))─  
    2 |
```

# LpNorm

Calculates the root of p-th power of the sum of the absolute values of a tuple in the power of p.

## Syntax

```
LpNorm(tuple, p)
```

Alias: `normLp`.

## Arguments

- `tuple` — **Tuple**.
- `p` — The power. Possible values: real number in [1; inf). **UInt** or **Float**.

## Returned value

- **Lp-norm**

Type: **Float**.

## Example

Query:

```
SELECT LpNorm((1, -2), 2);
```

Result:

```
└─LpNorm((1, -2), 2)─  
    2.23606797749979 |
```

# L1Distance

Calculates the distance between two points (the values of the tuples are the coordinates) in L1 space (1-norm (**taxicab geometry** distance)).

## Syntax

```
L1Distance(tuple1, tuple2)
```

Alias: `distanceL1`.

## Arguments

- `tuple1` — First tuple. [Tuple](#).
- `tuple1` — Second tuple. [Tuple](#).

## Returned value

- 1-norm distance.

Type: [Float](#).

## Example

Query:

```
SELECT L1Distance((1, 2), (2, 3));
```

Result:

```
L1Distance((1, 2), (2, 3))—  
2 |
```

# L2Distance

Calculates the distance between two points (the values of the tuples are the coordinates) in Euclidean space ([Euclidean distance](#)).

## Syntax

```
L2Distance(tuple1, tuple2)
```

Alias: `distanceL2`.

## Arguments

- `tuple1` — First tuple. [Tuple](#).
- `tuple1` — Second tuple. [Tuple](#).

## Returned value

- 2-norm distance.

Type: [Float](#).

## Example

Query:

```
SELECT L2Distance((1, 2), (2, 3));
```

Result:

```
L2Distance((1, 2), (2, 3))  
1.4142135623730951 |
```

## LinfDistance

Calculates the distance between two points (the values of the tuples are the coordinates) in  $L_{\infty}$  space (**maximum norm**).

### Syntax

```
LinfDistance(tuple1, tuple2)
```

Alias: `distanceLinf`.

### Arguments

- `tuple1` — First tuple. **Tuple**.
- `tuple1` — Second tuple. **Tuple**.

### Returned value

- Infinity-norm distance.

Type: **Float**.

### Example

Query:

```
SELECT LinfDistance((1, 2), (2, 3));
```

Result:

```
LinfDistance((1, 2), (2, 3))  
1 |
```

## LpDistance

Calculates the distance between two points (the values of the tuples are the coordinates) in  $L_p$  space (**p-norm distance**).

### Syntax

```
LpDistance(tuple1, tuple2, p)
```

Alias: `distanceLp`.

### Arguments

- `tuple1` — First tuple. **Tuple**.
- `tuple1` — Second tuple. **Tuple**.
- `p` — The power. Possible values: real number from  $[1; \infty)$ . **UInt** or **Float**.

## Returned value

- p-norm distance.

Type: **Float**.

## Example

Query:

```
SELECT LpDistance((1, 2), (2, 3), 3);
```

Result:

```
└─LpDistance((1, 2), (2, 3), 3)─  
    1.2599210498948732 |
```

## L1Normalize

Calculates the unit vector of a given vector (the values of the tuple are the coordinates) in **L1** space (**taxicab geometry**).

## Syntax

```
L1Normalize(tuple)
```

Alias: **normalizeL1**.

## Arguments

- **tuple** — **Tuple**.

## Returned value

- Unit vector.

Type: **Tuple of Float**.

## Example

Query:

```
SELECT L1Normalize((1, 2));
```

Result:

```
└─L1Normalize((1, 2))—————  
    (0.3333333333333333,0.6666666666666666) |
```

## L2Normalize

Calculates the unit vector of a given vector (the values of the tuple are the coordinates) in Euclidean space (using **Euclidean distance**).

## Syntax

```
L2Normalize(tuple)
```

Alias: `normalizeL1`.

## Arguments

- `tuple` — [Tuple](#).

## Returned value

- Unit vector.

Type: [Tuple of Float](#).

## Example

Query:

```
SELECT L2Normalize((3, 4));
```

Result:

```
L2Normalize((3, 4))  
|  
(0.6,0.8)
```

# LinfNormalize

Calculates the unit vector of a given vector (the values of the tuple are the coordinates) in  $L_{\infty}$  space (using [maximum norm](#)).

## Syntax

```
LinfNormalize(tuple)
```

Alias: `normalizeLinf`.

## Arguments

- `tuple` — [Tuple](#).

## Returned value

- Unit vector.

Type: [Tuple of Float](#).

## Example

Query:

```
SELECT LinfNormalize((3, 4));
```

Result:

```
└─LinfNormalize((3, 4)) ─  
  (0.75,1) └
```

## LpNormalize

Calculates the unit vector of a given vector (the values of the tuple are the coordinates) in L<sub>p</sub> space (using p-norm).

### Syntax

```
LpNormalize(tuple, p)
```

Alias: `normalizeLp`.

### Arguments

- `tuple` — [Tuple](#).
- `p` — The power. Possible values: any number from [1;inf). [UInt](#) or [Float](#).

### Returned value

- Unit vector.

Type: [Tuple of Float](#).

### Example

Query:

```
SELECT LpNormalize((3, 4),5);
```

Result:

```
└─LpNormalize((3, 4), 5) ─  
  (0.7187302630182624,0.9583070173576831) └
```

## cosineDistance

Calculates the cosine distance between two vectors (the values of the tuples are the coordinates). The less the returned value is, the more similar are the vectors.

### Syntax

```
cosineDistance(tuple1, tuple2)
```

### Arguments

- `tuple1` — First tuple. [Tuple](#).
- `tuple2` — Second tuple. [Tuple](#).

### Returned value

- Cosine of the angle between two vectors substracted from one.

Type: **Float**.

## Example

Query:

```
SELECT cosineDistance((1, 2), (2, 3));
```

Result:

```
cosineDistance((1, 2), (2, 3))—  
0.007722123286332261 |
```

# Encryption functions

These functions implement encryption and decryption of data with AES (Advanced Encryption Standard) algorithm.

Key length depends on encryption mode. It is 16, 24, and 32 bytes long for `-128-`, `-196-`, and `-256-` modes respectively.

Initialization vector length is always 16 bytes (bytes in excess of 16 are ignored).

Note that these functions work slowly until ClickHouse 21.1.

## encrypt

This function encrypts data using these modes:

- `aes-128-ecb`, `aes-192-ecb`, `aes-256-ecb`
- `aes-128-cbc`, `aes-192-cbc`, `aes-256-cbc`
- `aes-128-cfb1`, `aes-192-cfb1`, `aes-256-cfb1`
- `aes-128-cfb8`, `aes-192-cfb8`, `aes-256-cfb8`
- `aes-128-cfb128`, `aes-192-cfb128`, `aes-256-cfb128`
- `aes-128-ofb`, `aes-192-ofb`, `aes-256-ofb`
- `aes-128-gcm`, `aes-192-gcm`, `aes-256-gcm`

## Syntax

```
encrypt('mode', 'plaintext', 'key' [, iv, aad])
```

## Arguments

- `mode` — Encryption mode. **String**.
- `plaintext` — Text that needs to be encrypted. **String**.
- `key` — Encryption key. **String**.
- `iv` — Initialization vector. Required for `-gcm` modes, optional for others. **String**.

- `aad` — Additional authenticated data. It isn't encrypted, but it affects decryption. Works only in `-gcm` modes, for others would throw an exception. **String**.

## Returned value

- Ciphertext binary string. **String**.

## Examples

Create this table:

Query:

```
CREATE TABLE encryption_test
(
    `comment` String,
    `secret` String
)
ENGINE = Memory;
```

Insert some data (please avoid storing the keys/ivs in the database as this undermines the whole concept of encryption), also storing 'hints' is unsafe too and used only for illustrative purposes:

Query:

```
INSERT INTO encryption_test VALUES('aes-256-cfb128 no IV', encrypt('aes-256-cfb128', 'Secret', '12345678910121314151617181920212'),\n('aes-256-cfb128 no IV, different key', encrypt('aes-256-cfb128', 'Secret', 'keykeykeykeykeykeykeykeykeyke')),\n('aes-256-cfb128 with IV', encrypt('aes-256-cfb128', 'Secret', '12345678910121314151617181920212', 'iviviviviviviv')),\n('aes-256-cbc no IV', encrypt('aes-256-cbc', 'Secret', '12345678910121314151617181920212'));
```

Query:

```
SELECT comment, hex(secret) FROM encryption_test;
```

Result:

comment	hex(secret)
aes-256-cfb128 no IV	B4972BDC4459
aes-256-cfb128 no IV, different key	2FF57C092DC9
aes-256-cfb128 with IV	5E6CB398F653
aes-256-cbc no IV	1BC0629A92450D9E73A00E7D02CF4142

Example with `-gcm`:

Query:

```
INSERT INTO encryption_test VALUES('aes-256-gcm', encrypt('aes-256-gcm', 'Secret', '12345678910121314151617181920212', 'iviviviviviviv'),\n('aes-256-gcm with AAD', encrypt('aes-256-gcm', 'Secret', '12345678910121314151617181920212', 'iviviviviviviv', 'aad'));
```

```
SELECT comment, hex(secret) FROM encryption_test WHERE comment LIKE '%gcm%';
```

Result:

comment	hex(secret)
aes-256-gcm	A8A3CCBC6426CFEEB60E4EAE03D3E94204C1B09E0254
aes-256-gcm with AAD	A8A3CCBC6426D9A1017A0A932322F1852260A4AD6837

## aes\_encrypt\_mysql

Compatible with mysql encryption and resulting ciphertext can be decrypted with [AES\\_DECRYPT](#) function.

Will produce the same ciphertext as `encrypt` on equal inputs. But when `key` or `iv` are longer than they should normally be, `aes_encrypt_mysql` will stick to what MySQL's `aes_encrypt` does: 'fold' `key` and ignore excess bits of `iv`.

Supported encryption modes:

- aes-128-ecb, aes-192-ecb, aes-256-ecb
- aes-128-cbc, aes-192-cbc, aes-256-cbc
- aes-128-cfb1, aes-192-cfb1, aes-256-cfb1
- aes-128-cfb8, aes-192-cfb8, aes-256-cfb8
- aes-128-cfb128, aes-192-cfb128, aes-256-cfb128
- aes-128-ofb, aes-192-ofb, aes-256-ofb

### Syntax

```
aes_encrypt_mysql('mode', 'plaintext', 'key' [, iv])
```

### Arguments

- `mode` — Encryption mode. [String](#).
- `plaintext` — Text that needs to be encrypted. [String](#).
- `key` — Encryption key. If key is longer than required by mode, MySQL-specific key folding is performed. [String](#).
- `iv` — Initialization vector. Optional, only first 16 bytes are taken into account [String](#).

### Returned value

- Ciphertext binary string. [String](#).

### Examples

Given equal input `encrypt` and `aes_encrypt_mysql` produce the same ciphertext:

Query:

```
SELECT encrypt('aes-256-cfb128', 'Secret', '12345678910121314151617181920212', 'iviviviviviviv') =
aes_encrypt_mysql('aes-256-cfb128', 'Secret', '12345678910121314151617181920212', 'iviviviviviviv') AS
ciphertexts_equal;
```

Result:

```
ciphertexts_equal  
1
```

But `encrypt` fails when `key` or `iv` is longer than expected:

Query:

```
SELECT encrypt('aes-256-cfb128', 'Secret', '123456789101213141516171819202122', 'iviviviviviviv123');
```

Result:

```
Received exception from server (version 21.1.2):  
Code: 36. DB::Exception: Received from localhost:9000. DB::Exception: Invalid key size: 33 expected 32: While  
processing encrypt('aes-256-cfb128', 'Secret', '123456789101213141516171819202122', 'iviviviviviviv123').
```

While `aes_encrypt_mysql` produces MySQL-compatitalbe output:

Query:

```
SELECT hex(aes_encrypt_mysql('aes-256-cfb128', 'Secret', '123456789101213141516171819202122',  
'iviviviviviviv123')) AS ciphertext;
```

Result:

```
ciphertext  
24E9E4966469
```

Notice how supplying even longer IV produces the same result

Query:

```
SELECT hex(aes_encrypt_mysql('aes-256-cfb128', 'Secret', '123456789101213141516171819202122',  
'iviviviviviviv123456')) AS ciphertext
```

Result:

```
ciphertext  
24E9E4966469
```

Which is binary equal to what MySQL produces on same inputs:

```
mysql> SET block_encryption_mode='aes-256-cfb128';  
Query OK, 0 rows affected (0.00 sec)  
  
mysql> SELECT aes_encrypt('Secret', '123456789101213141516171819202122', 'iviviviviviviv123456') as  
ciphertext;  
+-----+  
| ciphertext |  
+-----+  
| 0x24E9E4966469 |  
+-----+  
1 row in set (0.00 sec)
```

## decrypt

This function decrypts ciphertext into a plaintext using these modes:

- aes-128-ecb, aes-192-ecb, aes-256-ecb
- aes-128-cbc, aes-192-cbc, aes-256-cbc
- aes-128-cfb1, aes-192-cfb1, aes-256-cfb1
- aes-128-cfb8, aes-192-cfb8, aes-256-cfb8
- aes-128-cfb128, aes-192-cfb128, aes-256-cfb128
- aes-128-ofb, aes-192-ofb, aes-256-ofb
- aes-128-gcm, aes-192-gcm, aes-256-gcm

## Syntax

```
decrypt('mode', 'ciphertext', 'key' [, iv, aad])
```

## Arguments

- `mode` — Decryption mode. [String](#).
- `ciphertext` — Encrypted text that needs to be decrypted. [String](#).
- `key` — Decryption key. [String](#).
- `iv` — Initialization vector. Required for `-gcm` modes, optional for others. [String](#).
- `aad` — Additional authenticated data. Won't decrypt if this value is incorrect. Works only in `-gcm` modes, for others would throw an exception. [String](#).

## Returned value

- Decrypted String. [String](#).

## Examples

Re-using table from [encrypt](#).

Query:

```
SELECT comment, hex(secret) FROM encryption_test;
```

Result:

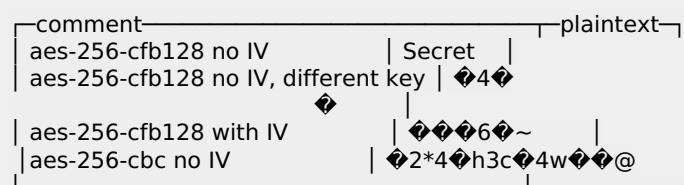
comment	hex(secret)
aes-256-gcm	A8A3CCBC6426CFEEB60E4EAE03D3E94204C1B09E0254
aes-256-gcm with AAD	A8A3CCBC6426D9A1017A0A932322F1852260A4AD6837
comment	hex(secret)
aes-256-cfb128 no IV	B4972BDC4459
aes-256-cfb128 no IV, different key	2FF57C092DC9
aes-256-cfb128 with IV	5E6CB398F653
aes-256-cbc no IV	1BC0629A92450D9E73A00E7D02CF4142

Now let's try to decrypt all that data.

Query:

```
SELECT comment, decrypt('aes-256-cfb128', secret, '12345678910121314151617181920212') as plaintext FROM encryption_test
```

Result:



Notice how only a portion of the data was properly decrypted, and the rest is gibberish since either mode, key, or iv were different upon encryption.

## aes\_decrypt\_mysql

Compatible with mysql encryption and decrypts data encrypted with [AES\\_ENCRYPT](#) function.

Will produce same plaintext as `decrypt` on equal inputs. But when `key` or `iv` are longer than they should normally be, `aes_decrypt_mysql` will stick to what MySQL's `aes_decrypt` does: 'fold' `key` and ignore excess bits of `IV`.

Supported decryption modes:

- aes-128-ecb, aes-192-ecb, aes-256-ecb
- aes-128-cbc, aes-192-cbc, aes-256-cbc
- aes-128-cfb1, aes-192-cfb1, aes-256-cfb1
- aes-128-cfb8, aes-192-cfb8, aes-256-cfb8
- aes-128-cfb128, aes-192-cfb128, aes-256-cfb128
- aes-128-ofb, aes-192-ofb, aes-256-ofb

## Syntax

```
aes_decrypt_mysql('mode', 'ciphertext', 'key' [, iv])
```

## Arguments

- `mode` — Decryption mode. [String](#).
- `ciphertext` — Encrypted text that needs to be decrypted. [String](#).
- `key` — Decryption key. [String](#).
- `iv` — Initialization vector. Optional. [String](#).

## Returned value

- Decrypted String. [String](#).

## Examples

Let's decrypt data we've previously encrypted with MySQL:

```
mysql> SET block_encryption_mode='aes-256-cfb128';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT aes_encrypt('Secret', '123456789101213141516171819202122', 'iviviviviviviv123456') as ciphertext;
+-----+
| ciphertext |
+-----+
| 0x24E9E4966469 |
+-----+
1 row in set (0.00 sec)
```

Query:

```
SELECT aes_decrypt_mysql('aes-256-cfb128', unhex('24E9E4966469'), '123456789101213141516171819202122',
'iviviviviviv123456') AS plaintext
```

Result:

plaintext	
Secret	

## [experimental] Natural Language Processing functions

### Warning

This is an experimental feature that is currently in development and is not ready for general use. It will change in unpredictable backwards-incompatible ways in future releases. Set `allow_experimental_nlp_functions = 1` to enable it.

## stem

Performs stemming on a given word.

### Syntax

```
stem('language', word)
```

### Arguments

- `language` — Language which rules will be applied. Must be in lowercase. **String**.
- `word` — word that needs to be stemmed. Must be in lowercase. **String**.

### Examples

Query:

```
SELECT arrayMap(x -> stem('en', x), ['I', 'think', 'it', 'is', 'a', 'blessing', 'in', 'disguise']) as res;
```

Result:

```
res
['I', 'think', 'it', 'is', 'a', 'bless', 'in', 'disguis'] |
```

## lemmatize

Performs lemmatization on a given word. Needs dictionaries to operate, which can be obtained [here](#).

### Syntax

```
lemmatize('language', word)
```

### Arguments

- language — Language which rules will be applied. [String](#).
- word — Word that needs to be lemmatized. Must be lowercase. [String](#).

### Examples

Query:

```
SELECT lemmatize('en', 'wolves');
```

Result:

```
lemmatize("wolves")|
  "wolf" |
```

Configuration:

```
<lemmatizers>
  <lemmatizer>
    <lang>en</lang>
    <path>en.bin</path>
  </lemmatizer>
</lemmatizers>
```

## synonyms

Finds synonyms to a given word. There are two types of synonym extensions: `plain` and `wordnet`.

With the `plain` extension type we need to provide a path to a simple text file, where each line corresponds to a certain synonym set. Words in this line must be separated with space or tab characters.

With the `wordnet` extension type we need to provide a path to a directory with WordNet thesaurus in it. Thesaurus must contain a WordNet sense index.

### Syntax

```
synonyms('extension_name', word)
```

### Arguments

- `extension_name` — Name of the extension in which search will be performed. [String](#).

- `word` — Word that will be searched in extension. [String](#).

## Examples

Query:

```
SELECT synonyms('list', 'important');
```

Result:

```
synonyms('list', 'important')  
['important','big','critical','crucial'] |
```

Configuration:

```
<synonyms_extensions>  
  <extension>  
    <name>en</name>  
    <type>plain</type>  
    <path>en.txt</path>  
  </extension>  
  <extension>  
    <name>en</name>  
    <type>wordnet</type>  
    <path>en/</path>  
  </extension>  
</synonyms_extensions>
```

# Other Functions

## hostName()

Returns a string with the name of the host that this function was performed on. For distributed processing, this is the name of the remote server host, if the function is performed on a remote server. If it is executed in the context of a distributed table, then it generates a normal column with values relevant to each shard. Otherwise it produces a constant value.

## getMacro

Gets a named value from the [macros](#) section of the server configuration.

### Syntax

```
getMacro(name);
```

### Arguments

- `name` — Name to retrieve from the `macros` section. [String](#).

### Returned value

- Value of the specified macro.

Type: [String](#).

### Example

The example `macros` section in the server configuration file:

```
<macros>
  <test>Value</test>
</macros>
```

Query:

```
SELECT getMacro('test');
```

Result:

```
getMacro('test')
Value |
```

An alternative way to get the same value:

```
SELECT * FROM system.macros
WHERE macro = 'test';
```

```
macro substitution
test | Value |
```

## FQDN

Returns the fully qualified domain name.

### Syntax

```
fqdn();
```

This function is case-insensitive.

### Returned value

- String with the fully qualified domain name.

Type: String.

### Example

Query:

```
SELECT FQDN();
```

Result:

```
FQDN()
clickhouse.ru-central1.internal |
```

## basename

Extracts the trailing part of a string after the last slash or backslash. This function is often used to extract the filename from a path.

```
basename( expr )
```

## Arguments

- `expr` — Expression resulting in a **String** type value. All the backslashes must be escaped in the resulting value.

## Returned Value

A string that contains:

- The trailing part of a string after the last slash or backslash.

If the input string contains a path ending with slash or backslash, for example, `/` or `c:\`, the function returns an empty string.

- The original string if there are no slashes or backslashes.

## Example

```
SELECT 'some/long/path/to/file' AS a, basename(a)
```

a some\long\path\to\file | basename('some\\long\\path\\to\\file') |

```
SELECT 'some\\long\\path\\to\\file' AS a, basename(a)
```

a some\long\path\to\file | basename('some\\long\\path\\to\\file') |

```
SELECT 'some-file-name' AS a, basename(a)
```

a some-file-name | basename('some-file-name') |

## visibleWidth(x)

Calculates the approximate width when outputting values to the console in text format (tab-separated). This function is used by the system for implementing Pretty formats.

`NULL` is represented as a string corresponding to `NULL` in Pretty formats.

```
SELECT visibleWidth(NULL)
```

```
visibleWidth(NULL)─  
 4 |
```

## toTypeName(x)

Returns a string containing the type name of the passed argument.

If `NULL` is passed to the function as input, then it returns the `Nullable(Nothing)` type, which corresponds to an internal `NUL` representation in ClickHouse.

## blockSize()

Gets the size of the block.

In ClickHouse, queries are always run on blocks (sets of column parts). This function allows getting the size of the block that you called it for.

## byteSize

Returns estimation of uncompressed byte size of its arguments in memory.

### Syntax

```
byteSize(argument [, ...])
```

### Arguments

- `argument` — Value.

### Returned value

- Estimation of byte size of the arguments in memory.

Type: `UInt64`.

### Examples

For `String` arguments the function returns the string length + 9 (terminating zero + length).

Query:

```
SELECT byteSize('string');
```

Result:

```
byteSize('string')─  
 15 |
```

Query:

```

CREATE TABLE test
(
    `key` Int32,
    `u8` UInt8,
    `u16` UInt16,
    `u32` UInt32,
    `u64` UInt64,
    `i8` Int8,
    `i16` Int16,
    `i32` Int32,
    `i64` Int64,
    `f32` Float32,
    `f64` Float64
)
ENGINE = MergeTree
ORDER BY key;

INSERT INTO test VALUES(1, 8, 16, 32, 64, -8, -16, -32, -64, 32.32, 64.64);

SELECT key, byteSize(u8) AS `byteSize(UInt8)`, byteSize(u16) AS `byteSize(UInt16)`, byteSize(u32) AS `byteSize(UInt32)`, byteSize(u64) AS `byteSize(UInt64)`, byteSize(i8) AS `byteSize(Int8)`, byteSize(i16) AS `byteSize(Int16)`, byteSize(i32) AS `byteSize(Int32)`, byteSize(i64) AS `byteSize(Int64)`, byteSize(f32) AS `byteSize(Float32)`, byteSize(f64) AS `byteSize(Float64)` FROM test ORDER BY key ASC FORMAT Vertical;

```

Result:

```

Row 1:
_____
key:      1
byteSize(UInt8): 1
byteSize(UInt16): 2
byteSize(UInt32): 4
byteSize(UInt64): 8
byteSize(Int8): 1
byteSize(Int16): 2
byteSize(Int32): 4
byteSize(Int64): 8
byteSize(Float32): 4
byteSize(Float64): 8

```

If the function takes multiple arguments, it returns their combined byte size.

Query:

```
SELECT byteSize(NULL, 1, 0.3, "");
```

Result:

byteSize(NULL, 1, 0.3, "")
19

## materialize(x)

Turns a constant into a full column containing just one value.

In ClickHouse, full columns and constants are represented differently in memory. Functions work differently for constant arguments and normal arguments (different code is executed), although the result is almost always the same. This function is for debugging this behavior.

## ignore(...)

Accepts any arguments, including `NULL`. Always returns 0.

However, the argument is still evaluated. This can be used for benchmarks.

## sleep(seconds)

Sleeps 'seconds' seconds on each data block. You can specify an integer or a floating-point number.

## sleepEachRow(seconds)

Sleeps 'seconds' seconds on each row. You can specify an integer or a floating-point number.

## currentDatabase()

Returns the name of the current database.

You can use this function in table engine parameters in a CREATE TABLE query where you need to specify the database.

## currentUser()

Returns the login of current user. Login of user, that initiated query, will be returned in case distributed query.

```
SELECT currentUser();
```

Alias: `user()`, `USER()`.

### Returned values

- Login of current user.
- Login of user that initiated query in case of distributed query.

Type: String.

### Example

Query:

```
SELECT currentUser();
```

Result:

```
currentUser()  
default |
```

## isConstant

Checks whether the argument is a constant expression.

A constant expression means an expression whose resulting value is known at the query analysis (i.e. before execution). For example, expressions over [literals](#) are constant expressions.

The function is intended for development, debugging and demonstration.

### Syntax

```
isConstant(x)
```

### Arguments

- $x$  — Expression to check.

## Returned values

- 1 —  $x$  is constant.
- 0 —  $x$  is non-constant.

Type: [UInt8](#).

## Examples

Query:

```
SELECT isConstant(x + 1) FROM (SELECT 43 AS x)
```

Result:

```
isConstant(plus(x, 1))  
 1 |
```

Query:

```
WITH 3.14 AS pi SELECT isConstant(cos(pi))
```

Result:

```
isConstant(cos(pi))  
 1 |
```

Query:

```
SELECT isConstant(number) FROM numbers(1)
```

Result:

```
isConstant(number)  
 0 |
```

## isFinite( $x$ )

Accepts `Float32` and `Float64` and returns `UInt8` equal to 1 if the argument is not infinite and not a NaN, otherwise 0.

## isInfinite( $x$ )

Accepts `Float32` and `Float64` and returns `UInt8` equal to 1 if the argument is infinite, otherwise 0. Note that 0 is returned for a NaN.

## ifNotFinite

Checks whether floating point value is finite.

## Syntax

```
ifNotFinite(x,y)
```

## Arguments

- `x` — Value to be checked for infinity. Type: [Float\\*](#).
- `y` — Fallback value. Type: [Float\\*](#).

## Returned value

- `x` if `x` is finite.
- `y` if `x` is not finite.

## Example

Query:

```
SELECT 1/0 as infimum, ifNotFinite(infimum,42)
```

Result:

```
infimum---ifNotFinite(divide(1, 0), 42)---  
    inf |           42 |
```

You can get similar result by using [ternary operator](#): `isFinite(x) ? x : y`.

## isNaN(x)

Accepts `Float32` and `Float64` and returns `UInt8` equal to 1 if the argument is a `Nan`, otherwise 0.

**hasColumnInTable(['hostname'][, 'username'][, 'password']), [, 'database', 'table', 'column'])**

Accepts constant strings: database name, table name, and column name. Returns a `UInt8` constant expression equal to 1 if there is a column, otherwise 0. If the `hostname` parameter is set, the test will run on a remote server.

The function throws an exception if the table does not exist.

For elements in a nested data structure, the function checks for the existence of a column. For the nested data structure itself, the function returns 0.

## bar

Allows building a unicode-art diagram.

```
bar(x, min, max, width) draws a band with a width proportional to (x - min) and equal to width characters when  
x = max.
```

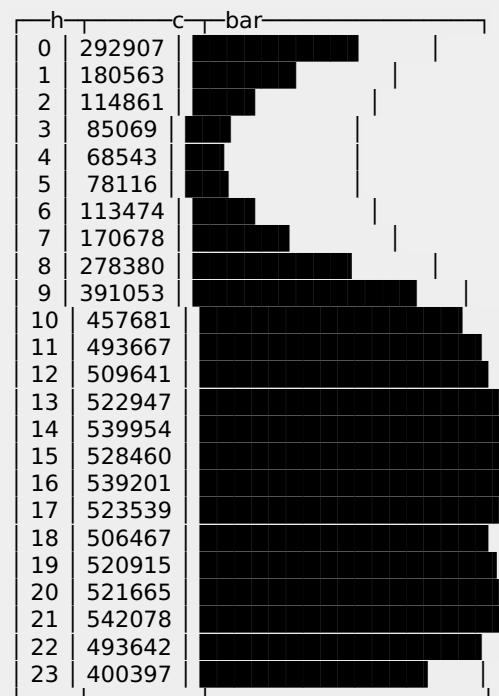
## Arguments

- `x` — Size to display.
- `min, max` — Integer constants. The value must fit in `Int64`.
- `width` — Constant, positive integer, can be fractional.

The band is drawn with accuracy to one eighth of a symbol.

Example:

```
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
```



## transform

Transforms a value according to the explicitly defined mapping of some elements to other ones.  
There are two variations of this function:

**transform(x, array\_from, array\_to, default)**

x – What to transform.

array\_from – Constant array of values for converting.

array\_to – Constant array of values to convert the values in ‘from’ to.

default – Which value to use if ‘x’ is not equal to any of the values in ‘from’.

array\_from and array\_to – Arrays of the same size.

Types:

`transform(T, Array(T), Array(U), U) -> U`

T and U can be numeric, string, or Date or DateTime types.

Where the same letter is indicated (T or U), for numeric types these might not be matching types, but types that have a common type.

For example, the first argument can have the Int64 type, while the second has the Array(UInt16) type.

If the 'x' value is equal to one of the elements in the 'array\_from' array, it returns the existing element (that is numbered the same) from the 'array\_to' array. Otherwise, it returns 'default'. If there are multiple matching elements in 'array\_from', it returns one of the matches.

Example:

```
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

## transform(x, array\_from, array\_to)

Differs from the first variation in that the 'default' argument is omitted.

If the 'x' value is equal to one of the elements in the 'array\_from' array, it returns the matching element (that is numbered the same) from the 'array\_to' array. Otherwise, it returns 'x'.

Types:

```
transform(T, Array(T), Array(T)) -> T
```

Example:

```
SELECT
    transform(domain(Rferer), ['yandex.ru', 'google.ru', 'vk.com'], ['www.yandex', 'example.com']) AS s,
    count() AS c
FROM test.hits
GROUP BY domain(Rferer)
ORDER BY count() DESC
LIMIT 10
```

s	c
www.yandex	2906259
[REDACTED].ru	867767
mail.yandex.ru	313599
[REDACTED].ru	107147
[REDACTED].ru	100355
news.yandex.ru	65040
[REDACTED].net	64515
example.com	59141
	57316

## formatReadableSize(x)

Accepts the size (number of bytes). Returns a rounded size with a suffix (KiB, MiB, etc.) as a string.

Example:

```
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

## formatReadableQuantity(x)

Accepts the number. Returns a rounded number with a suffix (thousand, million, billion, etc.) as a string.

It is useful for reading big numbers by human.

Example:

```
SELECT
    arrayJoin([1024, 1234 * 1000, (4567 * 1000) * 1000, 98765432101234]) AS number,
    formatReadableQuantity(number) AS number_for_humans
```

number	number_for_humans
1024	1.02 thousand
1234000	1.23 million
4567000000	4.57 billion
98765432101234	98.77 trillion

## formatReadableTimeDelta

Accepts the time delta in seconds. Returns a time delta with (year, month, day, hour, minute, second) as a string.

### Syntax

```
formatReadableTimeDelta(column[, maximum_unit])
```

### Arguments

- `column` — A column with numeric time delta.
- `maximum_unit` — Optional. Maximum unit to show. Acceptable values seconds, minutes, hours, days, months, years.

Example:

```
SELECT
    arrayJoin([100, 12345, 432546534]) AS elapsed,
    formatReadableTimeDelta(elapsed) AS time_delta
```

elapsed	time_delta
100	1 minute and 40 seconds
12345	3 hours, 25 minutes and 45 seconds
432546534	13 years, 8 months, 17 days, 7 hours, 48 minutes and 54 seconds

```
SELECT
    arrayJoin([100, 12345, 432546534]) AS elapsed,
    formatReadableTimeDelta(elapsed, 'minutes') AS time_delta
```

elapsed	time_delta
100	1 minute and 40 seconds
12345	205 minutes and 45 seconds
432546534	7209108 minutes and 54 seconds

## least(a, b)

Returns the smallest value from a and b.

## greatest(a, b)

Returns the largest value of a and b.

## uptime()

Returns the server's uptime in seconds.

If it is executed in the context of a distributed table, then it generates a normal column with values relevant to each shard. Otherwise it produces a constant value.

## version()

Returns the version of the server as a string.

If it is executed in the context of a distributed table, then it generates a normal column with values relevant to each shard. Otherwise it produces a constant value.

## buildId()

Returns the build ID generated by a compiler for the running ClickHouse server binary.

If it is executed in the context of a distributed table, then it generates a normal column with values relevant to each shard. Otherwise it produces a constant value.

## blockNumber

Returns the sequence number of the data block where the row is located.

## rowNumberInBlock

Returns the ordinal number of the row in the data block. Different data blocks are always recalculated.

## rowNumberInAllBlocks()

Returns the ordinal number of the row in the data block. This function only considers the affected data blocks.

## neighbor

The window function that provides access to a row at a specified offset which comes before or after the current row of a given column.

### Syntax

```
neighbor(column, offset[, default_value])
```

The result of the function depends on the affected data blocks and the order of data in the block.

### Warning

It can reach the neighbor rows only inside the currently processed data block.

The rows order used during the calculation of neighbor can differ from the order of rows returned to the user.

To prevent that you can make a subquery with **ORDER BY** and call the function from outside the subquery.

## Arguments

- `column` — A column name or scalar expression.
- `offset` — The number of rows forwards or backwards from the current row of `column`. [Int64](#).
- `default_value` — Optional. The value to be returned if offset goes beyond the scope of the block. Type of data blocks affected.

## Returned values

- Value for `column` in `offset` distance from current row if `offset` value is not outside block bounds.
- Default value for `column` if `offset` value is outside block bounds. If `default_value` is given, then it will be used.

Type: type of data blocks affected or default value type.

## Example

Query:

```
SELECT number, neighbor(number, 2) FROM system.numbers LIMIT 10;
```

Result:

number	neighbor(number, 2)
0	2
1	3
2	4
3	5
4	6
5	7
6	8
7	9
8	0
9	0

Query:

```
SELECT number, neighbor(number, 2, 999) FROM system.numbers LIMIT 10;
```

Result:

number	neighbor(number, 2, 999)
0	2
1	3
2	4
3	5
4	6
5	7
6	8
7	9
8	999
9	999

This function can be used to compute year-over-year metric value:

Query:

```
WITH toDate('2018-01-01') AS start_date
SELECT
    toStartOfMonth(start_date + (number * 32)) AS month,
    toInt32(month) % 100 AS money,
    neighbor(money, -12) AS prev_year,
    round(prev_year / money, 2) AS year_over_year
FROM numbers(16)
```

Result:

month	money	prev_year	year_over_year
2018-01-01	32	0	0
2018-02-01	63	0	0
2018-03-01	91	0	0
2018-04-01	22	0	0
2018-05-01	52	0	0
2018-06-01	83	0	0
2018-07-01	13	0	0
2018-08-01	44	0	0
2018-09-01	75	0	0
2018-10-01	5	0	0
2018-11-01	36	0	0
2018-12-01	66	0	0
2019-01-01	97	32	0.33
2019-02-01	28	63	2.25
2019-03-01	56	91	1.62
2019-04-01	87	22	0.25

## runningDifference(x)

Calculates the difference between successive row values in the data block.

Returns 0 for the first row and the difference from the previous row for each subsequent row.

### Warning

It can reach the previous row only inside the currently processed data block.

The result of the function depends on the affected data blocks and the order of data in the block.

The rows order used during the calculation of runningDifference can differ from the order of rows returned to the user.

To prevent that you can make a subquery with **ORDER BY** and call the function from outside the subquery.

Example:

```

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

Please note - block size affects the result. With each new block, the runningDifference state is reset.

```

SELECT
    number,
    runningDifference(number + 1) AS diff
FROM numbers(100000)
WHERE diff != 1

```

number	diff
0	0
65536	0

```
set max_block_size=100000 -- default value is 65536!
```

```

SELECT
    number,
    runningDifference(number + 1) AS diff
FROM numbers(100000)
WHERE diff != 1

```

number	diff
0	0

## runningDifferenceStartingWithFirstValue

Same as for [runningDifference](#), the difference is the value of the first row, returned the value of the first row, and each subsequent row returns the difference from the previous row.

## runningConcurrency

Calculates the number of concurrent events.

Each event has a start time and an end time. The start time is included in the event, while the end time is excluded. Columns with a start time and an end time must be of the same data type.

The function calculates the total number of active (concurrent) events for each event start time.

## Warning

Events must be ordered by the start time in ascending order. If this requirement is violated the function raises an exception.

Every data block is processed separately. If events from different data blocks overlap then they can not be processed correctly.

## Syntax

```
runningConcurrency(start, end)
```

## Arguments

- **start** — A column with the start time of events. [Date](#), [DateTime](#), or [DateTime64](#).
- **end** — A column with the end time of events. [Date](#), [DateTime](#), or [DateTime64](#).

## Returned values

- The number of concurrent events at each event start time.

Type: [UInt32](#)

## Example

Consider the table:

start	end
2021-03-03	2021-03-11
2021-03-06	2021-03-12
2021-03-07	2021-03-08
2021-03-11	2021-03-12

Query:

```
SELECT start, runningConcurrency(start, end) FROM example_table;
```

Result:

start	runningConcurrency(start, end)
2021-03-03	1
2021-03-06	2
2021-03-07	3
2021-03-11	2

## MACNumToString(num)

Accepts a [UInt64](#) number. Interprets it as a MAC address in big endian. Returns a string containing the corresponding MAC address in the format AA:BB:CC:DD:EE:FF (colon-separated numbers in hexadecimal form).

## MACStringToNum(s)

The inverse function of [MACNumToString](#). If the MAC address has an invalid format, it returns 0.

## MACStringToOUI(s)

Accepts a MAC address in the format AA:BB:CC:DD:EE:FF (colon-separated numbers in hexadecimal form). Returns the first three octets as a UInt64 number. If the MAC address has an invalid format, it returns 0.

## getSizeOfEnumType

Returns the number of fields in [Enum](#).

```
getSizeOfEnumType(value)
```

### Arguments:

- `value` — Value of type [Enum](#).

### Returned values

- The number of fields with [Enum](#) input values.
- An exception is thrown if the type is not [Enum](#).

### Example

```
SELECT getSizeOfEnumType( CAST('a' AS Enum8('a' = 1, 'b' = 2) ) ) AS x
```

```
x  
2 |
```

## blockSerializedSize

Returns size on disk (without taking into account compression).

```
blockSerializedSize(value[, value[, ...]])
```

### Arguments

- `value` — Any value.

### Returned values

- The number of bytes that will be written to disk for block of values (without compression).

### Example

Query:

```
SELECT blockSerializedSize(maxState(1)) as x
```

Result:

```
x  
2 |
```

## toColumnName

Returns the name of the class that represents the data type of the column in RAM.

```
toColumnName(value)
```

### Arguments:

- `value` — Any type of value.

### Returned values

- A string with the name of the class that is used for representing the `value` data type in RAM.

### Example of the difference between `toTypeName` and `' toColumnName`

```
SELECT toTypeName(CAST('2018-01-01 01:02:03' AS DateTime))
```

```
toTypeName(CAST('2018-01-01 01:02:03', 'DateTime'))  
|  
DateTime
```

```
SELECT toColumnName(CAST('2018-01-01 01:02:03' AS DateTime))
```

```
toColumnName(CAST('2018-01-01 01:02:03', 'DateTime'))  
|  
Const(UInt32)
```

The example shows that the `DateTime` data type is stored in memory as `Const(UInt32)`.

## dumpColumnStructure

Outputs a detailed description of data structures in RAM

```
dumpColumnStructure(value)
```

### Arguments:

- `value` — Any type of value.

### Returned values

- A string describing the structure that is used for representing the `value` data type in RAM.

### Example

```
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))
```

## defaultValueOfArgumentType

Outputs the default value for the data type.

Does not include default values for custom columns set by the user.

```
defaultValueOfArgumentType(expression)
```

### Arguments:

- `expression` — Arbitrary type of value or an expression that results in a value of an arbitrary type.

### Returned values

- `0` for numbers.
- Empty string for strings.
- `NULL` for **Nullable**.

### Example

```
SELECT defaultValueOfArgumentType( CAST(1 AS Int8) )
```

```
defaultValueOfArgumentType(CAST(1, 'Int8'))—  
0 |
```

```
SELECT defaultValueOfArgumentType( CAST(1 AS Nullable(Int8) ) )
```

```
defaultValueOfArgumentType(CAST(1, 'Nullable(Int8)'))—  
NULL |
```

## defaultValueOfTypeName

Outputs the default value for given type name.

Does not include default values for custom columns set by the user.

```
defaultValueOfTypeName(type)
```

### Arguments:

- `type` — A string representing a type name.

### Returned values

- `0` for numbers.
- Empty string for strings.
- `NULL` for **Nullable**.

### Example

```
SELECT defaultValueOfTypeName('Int8')
```

```
defaultValueOfTypeName('Int8')  
0 |
```

```
SELECT defaultValueOfTypeName('Nullable(Int8)')
```

```
defaultValueOfTypeName('Nullable(Int8)')  
NULL |
```

## indexHint

The function is intended for debugging and introspection purposes. The function ignores its argument and always returns 1. Arguments are not even evaluated.

But for the purpose of index analysis, the argument of this function is analyzed as if it was present directly without being wrapped inside `indexHint` function. This allows to select data in index ranges by the corresponding condition but without further filtering by this condition. The index in ClickHouse is sparse and using `indexHint` will yield more data than specifying the same condition directly.

### Syntax

```
SELECT * FROM table WHERE indexHint(<expression>)
```

### Returned value

1. Type: `UInt8`.

### Example

Here is the example of test data from the table `ontime`.

Input table:

```
SELECT count() FROM ontine
```

```
count()  
4276457 |
```

The table has indexes on the fields (`FlightDate`, `(Year, FlightDate)`).

Create a query, where the index is not used.

Query:

```
SELECT FlightDate AS k, count() FROM ontine GROUP BY k ORDER BY k
```

ClickHouse processed the entire table (Processed 4.28 million rows).

Result:

k	count()
2017-01-01	13970
2017-01-02	15882
.....	
2017-09-28	16411
2017-09-29	16384
2017-09-30	12520

To apply the index, select a specific date.

Query:

```
SELECT FlightDate AS k, count() FROM ontime WHERE k = '2017-09-15' GROUP BY k ORDER BY k
```

By using the index, ClickHouse processed a significantly smaller number of rows (Processed 32.74 thousand rows).

Result:

k	count()
2017-09-15	16428

Now wrap the expression `k = '2017-09-15'` into `indexHint` function.

Query:

```
SELECT
  FlightDate AS k,
  count()
FROM ontime
WHERE indexHint(k = '2017-09-15')
GROUP BY k
ORDER BY k ASC
```

ClickHouse used the index in the same way as the previous time (Processed 32.74 thousand rows).

The expression `k = '2017-09-15'` was not used when generating the result.

In example the `indexHint` function allows to see adjacent dates.

Result:

k	count()
2017-09-14	7071
2017-09-15	16428
2017-09-16	1077
2017-09-30	8167

## replicate

Creates an array with a single value.

Used for internal implementation of [arrayJoin](#).

```
SELECT replicate(x, arr);
```

### Arguments:

- `arr` — Original array. ClickHouse creates a new array of the same length as the original and fills it with the value `x`.
- `x` — The value that the resulting array will be filled with.

## Returned value

An array filled with the value `x`.

Type: `Array`.

## Example

Query:

```
SELECT replicate(1, ['a', 'b', 'c'])
```

Result:

```
replicate(1, ['a', 'b', 'c'])  
[1,1,1]
```

# filesystemAvailable

Returns amount of remaining space on the filesystem where the files of the databases located. It is always smaller than total free space ([filesystemFree](#)) because some space is reserved for OS.

## Syntax

```
filesystemAvailable()
```

## Returned value

- The amount of remaining space available in bytes.

Type: `UInt64`.

## Example

Query:

```
SELECT formatReadableSize(filesystemAvailable()) AS "Available space", toTypeName(filesystemAvailable()) AS  
"Type";
```

Result:

```
Available space Type  
30.75 GiB UInt64
```

# filesystemFree

Returns total amount of the free space on the filesystem where the files of the databases located. See also [filesystemAvailable](#)

## Syntax

```
filesystemFree()
```

## Returned value

- Amount of free space in bytes.

Type: [UInt64](#).

## Example

Query:

```
SELECT formatReadableSize(filesystemFree()) AS "Free space", toTypeName(filesystemFree()) AS "Type";
```

Result:

Free space	Type
32.39 GiB	UInt64

## filesystemCapacity

Returns the capacity of the filesystem in bytes. For evaluation, the [path](#) to the data directory must be configured.

## Syntax

```
filesystemCapacity()
```

## Returned value

- Capacity information of the filesystem in bytes.

Type: [UInt64](#).

## Example

Query:

```
SELECT formatReadableSize(filesystemCapacity()) AS "Capacity", toTypeName(filesystemCapacity()) AS "Type"
```

Result:

Capacity	Type
39.32 GiB	UInt64

## initializeAggregation

Calculates result of aggregate function based on single value. It is intended to use this function to initialize aggregate functions with combinator [-State](#). You can create states of aggregate functions and insert them to columns of type [AggregateFunction](#) or use initialized aggregates as default values.

## Syntax

```
initializeAggregation (aggregate_function, arg1, arg2, ..., argN)
```

## Arguments

- `aggregate_function` — Name of the aggregation function to initialize. [String](#).
- `arg` — Arguments of aggregate function.

## Returned value(s)

- Result of aggregation for every row passed to the function.

The return type is the same as the return type of function, that `initializeAggregation` takes as first argument.

## Example

Query:

```
SELECT uniqMerge(state) FROM (SELECT initializeAggregation('uniqState', number % 3) AS state FROM numbers(10000));
```

Result:

```
└─uniqMerge(state)─  
   3 |
```

Query:

```
SELECT finalizeAggregation(state), toTypeName(state) FROM (SELECT initializeAggregation('sumState', number % 3) AS state FROM numbers(5));
```

Result:

```
└─finalizeAggregation(state)─ └─toTypeName(state)─  
    0 | AggregateFunction(sum, UInt8) |  
    1 | AggregateFunction(sum, UInt8) |  
    2 | AggregateFunction(sum, UInt8) |  
    0 | AggregateFunction(sum, UInt8) |  
    1 | AggregateFunction(sum, UInt8) |
```

Example with `AggregatingMergeTree` table engine and `AggregateFunction` column:

```
CREATE TABLE metrics  
(  
    key UInt64,  
    value AggregateFunction(sum, UInt64) DEFAULT initializeAggregation('sumState', toUInt64(0))  
)  
ENGINE = AggregatingMergeTree  
ORDER BY key
```

```
INSERT INTO metrics VALUES (0, initializeAggregation('sumState', toUInt64(42)))
```

## See Also

- [arrayReduce](#)

# finalizeAggregation

Takes state of aggregate function. Returns result of aggregation (or finalized state when using [-State](#) combinator).

## Syntax

```
finalizeAggregation(state)
```

## Arguments

- `state` — State of aggregation. [AggregateFunction](#).

## Returned value(s)

- Value/values that was aggregated.

Type: Value of any types that was aggregated.

## Examples

Query:

```
SELECT finalizeAggregation(( SELECT countState(number) FROM numbers(10)));
```

Result:

```
finalizeAggregation(_subquery16)─  
10 |
```

Query:

```
SELECT finalizeAggregation(( SELECT sumState(number) FROM numbers(10)));
```

Result:

```
finalizeAggregation(_subquery20)─  
45 |
```

Note that `NULL` values are ignored.

Query:

```
SELECT finalizeAggregation(arrayReduce('anyState', [NULL, 2, 3]));
```

Result:

```
finalizeAggregation(arrayReduce('anyState', [NULL, 2, 3]))─  
2 |
```

Combined example:

Query:

```

WITH initializeAggregation('sumState', number) AS one_row_sum_state
SELECT
    number,
    finalizeAggregation(one_row_sum_state) AS one_row_sum,
    runningAccumulate(one_row_sum_state) AS cumulative_sum
FROM numbers(10);

```

Result:

number	one_row_sum	cumulative_sum
0	0	0
1	1	1
2	2	3
3	3	6
4	4	10
5	5	15
6	6	21
7	7	28
8	8	36
9	9	45

## See Also

- [arrayReduce](#)
- [initializeAggregation](#)

## runningAccumulate

Accumulates states of an aggregate function for each row of a data block.

### Warning

The state is reset for each new data block.

## Syntax

```
runningAccumulate(agg_state[, grouping]);
```

## Arguments

- `agg_state` — State of the aggregate function. [AggregateFunction](#).
- `grouping` — Grouping key. Optional. The state of the function is reset if the `grouping` value is changed. It can be any of the [supported data types](#) for which the equality operator is defined.

## Returned value

- Each resulting row contains a result of the aggregate function, accumulated for all the input rows from 0 to the current position. `runningAccumulate` resets states for each new data block or when the `grouping` value changes.

Type depends on the aggregate function used.

## Examples

Consider how you can use `runningAccumulate` to find the cumulative sum of numbers without and with grouping.

Query:

```
SELECT k, runningAccumulate(sum_k) AS res FROM (SELECT number as k, sumState(k) AS sum_k FROM numbers(10) GROUP BY k ORDER BY k);
```

Result:

k	res
0	0
1	1
2	3
3	6
4	10
5	15
6	21
7	28
8	36
9	45

The subquery generates `sumState` for every number from 0 to 9. `sumState` returns the state of the `sum` function that contains the sum of a single number.

The whole query does the following:

1. For the first row, `runningAccumulate` takes `sumState(0)` and returns 0.
2. For the second row, the function merges `sumState(0)` and `sumState(1)` resulting in `sumState(0 + 1)`, and returns 1 as a result.
3. For the third row, the function merges `sumState(0 + 1)` and `sumState(2)` resulting in `sumState(0 + 1 + 2)`, and returns 3 as a result.
4. The actions are repeated until the block ends.

The following example shows the grouping parameter usage:

Query:

```
SELECT
    grouping,
    item,
    runningAccumulate(state, grouping) AS res
FROM
(
    SELECT
        tolnt8(number / 4) AS grouping,
        number AS item,
        sumState(number) AS state
    FROM numbers(15)
    GROUP BY item
    ORDER BY item ASC
);
```

Result:

grouping	item	res
0	0	0
0	1	1
0	2	3
0	3	6
1	4	4
1	5	9
1	6	15
1	7	22
2	8	8
2	9	17
2	10	27
2	11	38
3	12	12
3	13	25
3	14	39

As you can see, `runningAccumulate` merges states for each group of rows separately.

## joinGet

The function lets you extract data from the table the same way as from a [dictionary](#).

Gets data from [Join](#) tables using the specified join key.

Only supports tables created with the `ENGINE = Join(ANY, LEFT, <join_keys>)` statement.

### Syntax

```
joinGet(join_storage_table_name, `value_column`, join_keys)
```

### Arguments

- `join_storage_table_name` — an [identifier](#) indicates where search is performed. The identifier is searched in the default database (see parameter `default_database` in the config file). To override the default database, use the `USE db_name` or specify the database and the table through the separator `db_name.db_table`, see the example.
- `value_column` — name of the column of the table that contains required data.
- `join_keys` — list of keys.

### Returned value

Returns list of values corresponded to list of keys.

If certain does not exist in source table then `0` or `null` will be returned based on `join_use_nulls` setting.

More info about `join_use_nulls` in [Join operation](#).

### Example

Input table:

```
CREATE DATABASE db_test
CREATE TABLE db_test.id_val(`id` UInt32, `val` UInt32) ENGINE = Join(ANY, LEFT, id) SETTINGS join_use_nulls = 1
INSERT INTO db_test.id_val VALUES (1,11)(2,12)(4,13)
```

id	val
4	13
2	12
1	11

Query:

```
SELECT joinGet(db_test.id_val,'val',toUInt32(number)) from numbers(4) SETTINGS join_use_nulls = 1
```

Result:

joinGet(db_test.id_val, 'val', toUInt32(number))
0
11
12
0

## modelEvaluate(model\_name, ...)

Evaluate external model.

Accepts a model name and model arguments. Returns Float64.

## throwIf(x[, custom\_message])

Throw an exception if the argument is non zero.

custom\_message - is an optional parameter: a constant string, provides an error message

```
SELECT throwIf(number = 3, 'Too many') FROM numbers(10);
```

```
↙ Progress: 0.00 rows, 0.00 B (0.00 rows/s., 0.00 B/s.) Received exception from server (version 19.14.1):
Code: 395. DB::Exception: Received from localhost:9000. DB::Exception: Too many.
```

## identity

Returns the same value that was used as its argument. Used for debugging and testing, allows to cancel using index, and get the query performance of a full scan. When query is analyzed for possible use of index, the analyzer does not look inside `identity` functions. Also constant folding is not applied too.

### Syntax

```
identity(x)
```

### Example

Query:

```
SELECT identity(42)
```

Result:

```
identity(42)─  
 42 |
```

## randomPrintableASCII

Generates a string with a random set of **ASCII** printable characters.

### Syntax

```
randomPrintableASCII(length)
```

### Arguments

- length — Resulting string length. Positive integer.

If you pass `length < 0` , behavior of the function is undefined.

### Returned value

- String with a random set of **ASCII** printable characters.

Type: **String**

### Example

```
SELECT number, randomPrintableASCII(30) as str, length(str) FROM system.numbers LIMIT 3
```

```
number ─ str ─────────────────── length(randomPrintableASCII(30)) ─  
0 | SuiCOSTvC0csfABSw=UcSzp2.`rv8x | 30 |  
1 | 1Ag NIJ &RCN:*>HVPG;PE-nO"SUFDF | 30 |  
2 | /"+<"wUTh:=Ljj Vm!c&hl*m#XTfzz | 30 |
```

## randomString

Generates a binary string of the specified length filled with random bytes (including zero bytes).

### Syntax

```
randomString(length)
```

### Arguments

- length — String length. Positive integer.

### Returned value

- String filled with random bytes.

Type: **String**.

### Example

Query:

```
SELECT randomString(30) AS str, length(str) AS len FROM numbers(2) FORMAT Vertical;
```

Result:

```
Row 1:  
str: 3 G : pT ?w ti k aV f6  
len: 30
```

```
Row 2:  
str: 9 ,] ^ ) ]?? 8  
len: 30
```

## See Also

- [generateRandom](#)
- [randomPrintableASCII](#)

## randomFixedString

Generates a binary string of the specified length filled with random bytes (including zero bytes).

### Syntax

```
randomFixedString(length);
```

### Arguments

- `length` — String length in bytes. [UInt64](#).

### Returned value(s)

- String filled with random bytes.

Type: [FixedString](#).

### Example

Query:

```
SELECT randomFixedString(13) as rnd, toTypeName(rnd)
```

Result:

rnd	toTypeName(randomFixedString(13))
j��h��H��Z��	FixedString(13)

## randomStringUTF8

Generates a random string of a specified length. Result string contains valid UTF-8 code points. The value of code points may be outside of the range of assigned Unicode.

### Syntax

```
randomStringUTF8(length);
```

## Arguments

- `length` — Required length of the resulting string in code points. `UInt64`.

## Returned value(s)

- UTF-8 random string.

Type: **String**.

## Example

### Query:

```
SELECT randomStringUTF8(13)
```

## Result:

## getSetting

Returns the current value of a **custom setting**.

## Syntax

```
getSetting('custom setting');
```

## Parameter

- custom setting = The setting name. **String**.

### Returned value

- The setting current value

## Example

```
SET custom_a = 123;  
SELECT getSetting('custom_a');
```

## Result

123

#### **See Also**

- #### ■ Custom Settings

## isDecimalOverflow

Checks whether the **Decimal** value is out of its (or specified) precision.

## Syntax

```
isDecimalOverflow(d, [p])
```

## Arguments

- `d` — value. [Decimal](#).
- `p` — precision. Optional. If omitted, the initial precision of the first argument is used. Using of this parameter could be helpful for data extraction to another DBMS or file. [UInt8](#).

## Returned values

- `1` — Decimal value has more digits than its precision allows.
- `0` — Decimal value satisfies the specified precision.

## Example

Query:

```
SELECT isDecimalOverflow(toDecimal32(1000000000, 0), 9),  
       isDecimalOverflow(toDecimal32(1000000000, 0)),  
       isDecimalOverflow(toDecimal32(-1000000000, 0), 9),  
       isDecimalOverflow(toDecimal32(-1000000000, 0));
```

Result:

```
1 1 1 1
```

## countDigits

Returns number of decimal digits you need to represent the value.

## Syntax

```
countDigits(x)
```

## Arguments

- `x` — [Int](#) or [Decimal](#) value.

## Returned value

Number of digits.

Type: [UInt8](#).

!!! note "Note"

For [Decimal](#) values takes into account their scales: calculates result over underlying integer type which is `(value * scale)`. For example: `countDigits(42) = 2`, `countDigits(42.000) = 5`, `countDigits(0.04200) = 4`. I.e. you may check decimal overflow for [Decimal64](#) with `countDecimal(x) > 18`. It's a slow variant of [isDecimalOverflow](#).

## Example

Query:

```
SELECT countDigits(toDecimal32(1, 9)), countDigits(toDecimal32(-1, 9)),
       countDigits(toDecimal64(1, 18)), countDigits(toDecimal64(-1, 18)),
       countDigits(toDecimal128(1, 38)), countDigits(toDecimal128(-1, 38));
```

Result:

```
10 10 19 19 39 39
```

## errorCodeToName

### Returned value

- Variable name for the error code.

Type: [LowCardinality\(String\)](#).

### Syntax

```
errorCodeToName(1)
```

Result:

```
UNSUPPORTED_METHOD
```

## tcpPort

Returns [native interface](#) TCP port number listened by this server.

If it is executed in the context of a distributed table, then it generates a normal column, otherwise it produces a constant value.

### Syntax

```
tcpPort()
```

### Arguments

- None.

### Returned value

- The TCP port number.

Type: [UInt16](#).

### Example

Query:

```
SELECT tcpPort();
```

Result:

```
tcpPort()
9000 |
```

## See Also

- [tcp\\_port](#)

## currentProfiles

Returns a list of the current [settings profiles](#) for the current user.

The command [SET PROFILE](#) could be used to change the current setting profile. If the command [SET PROFILE](#) was not used the function returns the profiles specified at the current user's definition (see [CREATE USER](#)).

### Syntax

```
currentProfiles()
```

#### Returned value

- List of the current user settings profiles.

Type: [Array\(String\)](#).

## enabledProfiles

Returns settings profiles, assigned to the current user both explicitly and implicitly. Explicitly assigned profiles are the same as returned by the [currentProfiles](#) function. Implicitly assigned profiles include parent profiles of other assigned profiles, profiles assigned via granted roles, profiles assigned via their own settings, and the main default profile (see the `default_profile` section in the main server configuration file).

### Syntax

```
enabledProfiles()
```

#### Returned value

- List of the enabled settings profiles.

Type: [Array\(String\)](#).

## defaultProfiles

Returns all the profiles specified at the current user's definition (see [CREATE USER](#) statement).

### Syntax

```
defaultProfiles()
```

#### Returned value

- List of the default settings profiles.

Type: [Array\(String\)](#).

## currentRoles

Returns the names of the roles which are current for the current user. The current roles can be changed by the [SET ROLE](#) statement. If the SET ROLE statement was not used, the function `currentRoles` returns the same as `defaultRoles`.

## Syntax

```
currentRoles()
```

### Returned value

- List of the current roles for the current user.

Type: [Array\(String\)](#).

## enabledRoles

Returns the names of the current roles and the roles, granted to some of the current roles.

## Syntax

```
enabledRoles()
```

### Returned value

- List of the enabled roles for the current user.

Type: [Array\(String\)](#).

## defaultRoles

Returns the names of the roles which are enabled by default for the current user when he logins. Initially these are all roles granted to the current user (see [GRANT](#)), but that can be changed with the [SET DEFAULT ROLE](#) statement.

## Syntax

```
defaultRoles()
```

### Returned value

- List of the default roles for the current user.

Type: [Array\(String\)](#).

## getServerPort

Returns the number of the server port. When the port is not used by the server, throws an exception.

## Syntax

```
getServerPort(port_name)
```

### Arguments

- `port_name` — The name of the server port. [String](#). Possible values:

- 'tcp\_port'
- 'tcp\_port\_secure'
- 'http\_port'
- 'https\_port'
- 'interserver\_http\_port'
- 'interserver\_https\_port'
- 'mysql\_port'
- 'postgresql\_port'
- 'grpc\_port'
- 'prometheus.port'

### Returned value

- The number of the server port.

Type: [UInt16](#).

### Example

Query:

```
SELECT getServerPort('tcp_port');
```

Result:

```
getServerPort('tcp_port')—  
9000
```

## queryID

Returns the ID of the current query. Other parameters of a query can be extracted from the [system.query\\_log](#) table via `query_id`.

In contrast to [initialQueryID](#) function, `queryID` can return different results on different shards (see example).

### Syntax

```
queryID()
```

### Returned value

- The ID of the current query.

Type: [String](#)

### Example

Query:

```
CREATE TABLE tmp (str String) ENGINE = Log;
INSERT INTO tmp (*) VALUES ('a');
SELECT count(DISTINCT t) FROM (SELECT queryID() AS t FROM remote('127.0.0.{1..3}', currentDatabase(), 'tmp')
GROUP BY queryID());
```

Result:

count()
3

## initialQueryID

Returns the ID of the initial current query. Other parameters of a query can be extracted from the [system.query\\_log](#) table via `initial_query_id`.

In contrast to [queryID](#) function, `initialQueryID` returns the same results on different shards (see example).

### Syntax

```
initialQueryID()
```

### Returned value

- The ID of the initial current query.

Type: [String](#)

### Example

Query:

```
CREATE TABLE tmp (str String) ENGINE = Log;
INSERT INTO tmp (*) VALUES ('a');
SELECT count(DISTINCT t) FROM (SELECT initialQueryID() AS t FROM remote('127.0.0.{1..3}', currentDatabase(), 'tmp')
GROUP BY queryID());
```

Result:

count()
1

## shardNum

Returns the index of a shard which processes a part of data for a distributed query. Indices are started from 1.

If a query is not distributed then constant value 0 is returned.

### Syntax

```
shardNum()
```

### Returned value

- Shard index or constant 0.

Type: [UInt32](#).

## Example

In the following example a configuration with two shards is used. The query is executed on the [system.one](#) table on every shard.

Query:

```
CREATE TABLE shard_num_example (dummy UInt8)
    ENGINE=Distributed(test_cluster_two_shards_localhost, system, one, dummy);
SELECT dummy, shardNum(), shardCount() FROM shard_num_example;
```

Result:

dummy	shardNum()	shardCount()
0	2	2
0	1	2

## See Also

- [Distributed Table Engine](#)

## shardCount

Returns the total number of shards for a distributed query.

If a query is not distributed then constant value 0 is returned.

## Syntax

```
shardCount()
```

## Returned value

- Total number of shards or 0.

Type: [UInt32](#).

## See Also

- [shardNum\(\)](#) function example also contains `shardCount()` function call.

## getOSKernelVersion

Returns a string with the current OS kernel version.

## Syntax

```
getOSKernelVersion()
```

## Arguments

- None.

## Returned value

- The current OS kernel version.

Type: **String**.

## Example

Query:

```
SELECT getOSKernelVersion();
```

Result:

```
getOSKernelVersion()—  
Linux 4.15.0-55-generic |
```

## Aggregate Functions

Aggregate functions work in the **normal** way as expected by database experts.

ClickHouse also supports:

- **Parametric aggregate functions**, which accept other parameters in addition to columns.
- **Combinators**, which change the behavior of aggregate functions.

## NULL Processing

During aggregation, all **NULLs** are skipped.

### Examples:

Consider this table:

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

Let's say you need to total the values in the **y** column:

```
SELECT sum(y) FROM t_null_big
```

```
sum(y)—  
7 |
```

Now you can use the **groupArray** function to create an array from the **y** column:

```
SELECT groupArray(y) FROM t_null_big
```

```
groupArray(y)—  
[2,2,3] |
```

`groupArray` does not include `NULL` in the resulting array.

# COUNT

Counts the number of rows or not-`NULL` values.

ClickHouse supports the following syntaxes for `count`:

- `count(expr)` or `COUNT(DISTINCT expr)`.
- `count()` or `COUNT(*)`. The `count()` syntax is ClickHouse-specific.

## Arguments

The function can take:

- Zero parameters.
- One **expression**.

## Returned value

- If the function is called without parameters it counts the number of rows.
- If the **expression** is passed, then the function counts how many times this expression returned not null. If the expression returns a **Nullable**-type value, then the result of `count` stays not **Nullable**. The function returns 0 if the expression returned `NULL` for all the rows.

In both cases the type of the returned value is **UInt64**.

## Details

ClickHouse supports the `COUNT(DISTINCT ...)` syntax. The behavior of this construction depends on the `count_distinctImplementation` setting. It defines which of the `uniq*` functions is used to perform the operation. The default is the `uniqExact` function.

The `SELECT count() FROM table` query is optimized by default using metadata from MergeTree. If you need to use row-level security, disable optimization using the `optimize_trivial_count_query` setting.

However `SELECT count(nullable_column) FROM table` query can be optimized by enabling the `optimize_functions_to_subcolumns` setting. With `optimize_functions_to_subcolumns = 1` the function reads only `null` subcolumn instead of reading and processing the whole column data. The query `SELECT count(n) FROM table` transforms to `SELECT sum(NOT n.null) FROM table`.

## Examples

Example 1:

```
SELECT count() FROM t
```

```
count()
5 |
```

Example 2:

```
SELECT name, value FROM system.settings WHERE name = 'count_distinctImplementation'
```

name	value
count_distinctImplementation	uniqExact

```
SELECT count(DISTINCT num) FROM t
```

```
uniqExact(num)  
3 |
```

This example shows that `count(DISTINCT num)` is performed by the `uniqExact` function according to the `count_distinctImplementation` setting value.

## min

Aggregate function that calculates the minimum across a group of values.

Example:

```
SELECT min(salary) FROM employees;
```

```
SELECT department, min(salary) FROM employees GROUP BY department;
```

If you need non-aggregate function to choose a minimum of two values, see `least`:

```
SELECT least(a, b) FROM table;
```

## max

Aggregate function that calculates the maximum across a group of values.

Example:

```
SELECT max(salary) FROM employees;
```

```
SELECT department, max(salary) FROM employees GROUP BY department;
```

If you need non-aggregate function to choose a maximum of two values, see `greatest`:

```
SELECT greatest(a, b) FROM table;
```

## sum

Calculates the sum. Only works for numbers.

## avg

Calculates the arithmetic mean.

## Syntax

```
avg(x)
```

## Arguments

- `x` — input values, must be [Integer](#), [Float](#), or [Decimal](#).

## Returned value

- The arithmetic mean, always as [Float64](#).
- `NaN` if the input parameter `x` is empty.

## Example

Query:

```
SELECT avg(x) FROM values('x Int8', 0, 1, 2, 3, 4, 5);
```

Result:

```
avg(x)  
2.5 |
```

## Example

Create a temp table:

Query:

```
CREATE table test (t UInt8) ENGINE = Memory;
```

Get the arithmetic mean:

Query:

```
SELECT avg(t) FROM test;
```

Result:

```
avg(x)  
nan |
```

## any

Selects the first encountered value.

The query can be executed in any order and even in a different order each time, so the result of this function is indeterminate.

To get a determinate result, you can use the ‘min’ or ‘max’ function instead of ‘any’.

In some cases, you can rely on the order of execution. This applies to cases when SELECT comes from a subquery that uses ORDER BY.

When a SELECT query has the GROUP BY clause or at least one aggregate function, ClickHouse (in contrast to MySQL) requires that all expressions in the SELECT, HAVING, and ORDER BY clauses be calculated from keys or from aggregate functions. In other words, each column selected from the table must be used either in keys or inside aggregate functions. To get behavior like in MySQL, you can put the other columns in the `any` aggregate function.

## stddevPop

The result is equal to the square root of `varPop`.

### Note

This function uses a numerically unstable algorithm. If you need **numerical stability** in calculations, use the `stddevPopStable` function. It works slower but provides a lower computational error.

## stddevSamp

The result is equal to the square root of `varSamp`.

### Note

This function uses a numerically unstable algorithm. If you need **numerical stability** in calculations, use the `stddevSampStable` function. It works slower but provides a lower computational error.

## varPop(x)

Calculates the amount  $\sum((x - \bar{x})^2) / n$ , where `n` is the sample size and  $\bar{x}$  is the average value of `x`.

In other words, dispersion for a set of values. Returns `Float64`.

### Note

This function uses a numerically unstable algorithm. If you need **numerical stability** in calculations, use the `varPopStable` function. It works slower but provides a lower computational error.

## varSamp

Calculates the amount  $\sum((x - \bar{x})^2) / (n - 1)$ , where `n` is the sample size and  $\bar{x}$  is the average value of `x`.

It represents an unbiased estimate of the variance of a random variable if passed values form its sample.

Returns `Float64`. When `n <= 1`, returns  $+\infty$ .

### Note

This function uses a numerically unstable algorithm. If you need **numerical stability** in calculations, use the `varSampStable` function. It works slower but provides a lower computational error.

## covarPop

Syntax: `covarPop(x, y)`

Calculates the value of  $\Sigma((x - \bar{x})(y - \bar{y})) / n$ .

### Note

This function uses a numerically unstable algorithm. If you need **numerical stability** in calculations, use the `covarPopStable` function. It works slower but provides a lower computational error.

## List of Aggregate Functions

Standard aggregate functions:

- [count](#)
- [min](#)
- [max](#)
- [sum](#)
- [avg](#)
- [any](#)
- [stddevPop](#)
- [stddevSamp](#)
- [varPop](#)
- [varSamp](#)
- [covarPop](#)
- [covarSamp](#)

ClickHouse-specific aggregate functions:

- [anyHeavy](#)
- [anyLast](#)
- [argMin](#)
- [argMax](#)
- [avgWeighted](#)
- [topK](#)
- [topKWeighted](#)

- groupArray
- groupUniqArray
- groupArrayInsertAt
- groupArrayMovingAvg
- groupArrayMovingSum
- groupBitAnd
- groupBitOr
- groupBitXor
- groupBitmap
- groupBitmapAnd
- groupBitmapOr
- groupBitmapXor
- sumWithOverflow
- sumMap
- minMap
- maxMap
- skewSamp
- skewPop
- kurtSamp
- kurtPop
- uniq
- uniqExact
- uniqCombined
- uniqCombined64
- uniqHLL12
- quantile
- quantiles
- quantileExact
- quantileExactLow
- quantileExactHigh
- quantileExactWeighted
- quantileTiming
- quantileTimingWeighted

- `quantileDeterministic`
  - `quantileTDigest`
  - `quantileTDigestWeighted`
  - `quantileBFloat16`
  - `quantileBFloat16Weighted`
  - `simpleLinearRegression`
  - `stochasticLinearRegression`
  - `stochasticLogisticRegression`
  - `categoricalInformationValue`
- 

## covarSamp

Calculates the value of  $\Sigma((x - \bar{x})(y - \bar{y})) / (n - 1)$

Returns `Float64`. When `n <= 1`, returns  $+\infty$ .

### Note

This function uses a numerically unstable algorithm. If you need **numerical stability** in calculations, use the `covarSampStable` function. It works slower but provides a lower computational error.

## anyHeavy

Selects a frequently occurring value using the **heavy hitters** algorithm. If there is a value that occurs more than in half the cases in each of the query's execution threads, this value is returned. Normally, the result is nondeterministic.

```
anyHeavy(column)
```

### Arguments

- `column` – The column name.

### Example

Take the `OnTime` data set and select any frequently occurring value in the `AirlineID` column.

```
SELECT anyHeavy(AirlineID) AS res  
FROM ontime
```

```
res-  
19690 |
```

## anyLast

Selects the last value encountered.

The result is just as indeterminate as for the [any](#) function.

## argMin

Calculates the `arg` value for a minimum `val` value. If there are several different values of `arg` for minimum values of `val`, returns the first of these values encountered.

### Syntax

```
argMin(arg, val)
```

### Arguments

- `arg` — Argument.
- `val` — Value.

### Returned value

- `arg` value that corresponds to minimum `val` value.

Type: matches `arg` type.

### Example

Input table:

user	salary
director	5000
manager	3000
worker	1000

Query:

```
SELECT argMin(user, salary) FROM salary
```

Result:

argMin(user, salary)
worker

## argMax

Calculates the `arg` value for a maximum `val` value. If there are several different values of `arg` for maximum values of `val`, returns the first of these values encountered.

### Syntax

```
argMax(arg, val)
```

### Arguments

- `arg` — Argument.

- `val` — Value.

## Returned value

- `arg` value that corresponds to maximum `val` value.

Type: matches `arg` type.

## Example

Input table:

user	salary
director	5000
manager	3000
worker	1000

Query:

```
SELECT argMax(user, salary) FROM salary;
```

Result:

```
argMax(user, salary)
director
```

# avgWeighted

Calculates the [weighted arithmetic mean](#).

## Syntax

```
avgWeighted(x, weight)
```

## Arguments

- `x` — Values.
- `weight` — Weights of the values.

`x` and `weight` must both be

[Integer](#),  
[floating-point](#), or  
[Decimal](#),

but may have different types.

## Returned value

- `Nan` if all the weights are equal to 0 or the supplied `weights` parameter is empty.
- Weighted mean otherwise.

**Return type** is always [Float64](#).

## Example

Query:

```
SELECT avgWeighted(x, w)
FROM values('x Int8, w Int8', (4, 1), (1, 0), (10, 2))
```

Result:

```
avgWeighted(x, weight)─
8 |
```

## Example

Query:

```
SELECT avgWeighted(x, w)
FROM values('x Int8, w Float64', (4, 1), (1, 0), (10, 2))
```

Result:

```
avgWeighted(x, weight)─
8 |
```

## Example

Query:

```
SELECT avgWeighted(x, w)
FROM values('x Int8, w Int8', (0, 0), (1, 0), (10, 0))
```

Result:

```
avgWeighted(x, weight)─
nan |
```

## Example

Query:

```
CREATE table test (t UInt8) ENGINE = Memory;
SELECT avgWeighted(t) FROM test
```

Result:

```
avgWeighted(x, weight)─
nan |
```

corr

Syntax: `corr(x, y)`

Calculates the Pearson correlation coefficient:  $\frac{\sum((x - \bar{x})(y - \bar{y}))}{\sqrt{\sum((x - \bar{x})^2) * \sum((y - \bar{y})^2)}}$

## Note

This function uses a numerically unstable algorithm. If you need **numerical stability** in calculations, use the `corrStable` function. It works slower but provides a lower computational error.

# topK

Returns an array of the approximately most frequent values in the specified column. The resulting array is sorted in descending order of approximate frequency of values (not by the values themselves).

Implements the **Filtered Space-Saving** algorithm for analyzing TopK, based on the reduce-and-combine algorithm from **Parallel Space Saving**.

```
topK(N)(column)
```

This function does not provide a guaranteed result. In certain situations, errors might occur and it might return frequent values that aren't the most frequent values.

We recommend using the `N < 10` value; performance is reduced with large `N` values. Maximum value of `N = 65536`.

### Arguments

- `N` – The number of elements to return.

If the parameter is omitted, default value 10 is used.

### Arguments

- `x` – The value to calculate frequency.

### Example

Take the **OnTime** data set and select the three most frequently occurring values in the `AirlineID` column.

```
SELECT topK(3)(AirlineID) AS res  
FROM ontime
```

```
res  
[19393,19790,19805] |
```

# topKWeighted

Returns an array of the approximately most frequent values in the specified column. The resulting array is sorted in descending order of approximate frequency of values (not by the values themselves). Additionally, the weight of the value is taken into account.

### Syntax

```
topKWeighted(N)(x, weight)
```

## Arguments

- `N` — The number of elements to return.
- `x` — The value.
- `weight` — The weight. Every value is accounted `weight` times for frequency calculation. **UInt64**.

## Returned value

Returns an array of the values with maximum approximate sum of weights.

## Example

Query:

```
SELECT topKWeighted(10)(number, number) FROM numbers(1000)
```

Result:

```
topKWeighted(10)(number, number)
[999,998,997,996,995,994,993,992,991,990] |
```

## See Also

- [topK](#)

# groupArray

Syntax: `groupArray(x)` or `groupArray(max_size)(x)`

Creates an array of argument values.

Values can be added to the array in any (indeterminate) order.

The second version (with the `max_size` parameter) limits the size of the resulting array to `max_size` elements. For example, `groupArray(1)(x)` is equivalent to `[any(x)]`.

In some cases, you can still rely on the order of execution. This applies to cases when `SELECT` comes from a subquery that uses `ORDER BY`.

# groupUniqArray

Syntax: `groupUniqArray(x)` or `groupUniqArray(max_size)(x)`

Creates an array from different argument values. Memory consumption is the same as for the [uniqExact](#) function.

The second version (with the `max_size` parameter) limits the size of the resulting array to `max_size` elements. For example, `groupUniqArray(1)(x)` is equivalent to `[any(x)]`.

# groupArrayInsertAt

Inserts a value into the array at the specified position.

## Syntax

```
groupArrayInsertAt(default_x, size)(x, pos)
```

If in one query several values are inserted into the same position, the function behaves in the following ways:

- If a query is executed in a single thread, the first one of the inserted values is used.
- If a query is executed in multiple threads, the resulting value is an undetermined one of the inserted values.

## Arguments

- `x` — Value to be inserted. [Expression](#) resulting in one of the [supported data types](#).
- `pos` — Position at which the specified element `x` is to be inserted. Index numbering in the array starts from zero. [UInt32](#).
- `default_x` — Default value for substituting in empty positions. Optional parameter. [Expression](#) resulting in the data type configured for the `x` parameter. If `default_x` is not defined, the [default values](#) are used.
- `size` — Length of the resulting array. Optional parameter. When using this parameter, the default value `default_x` must be specified. [UInt32](#).

## Returned value

- Array with inserted values.

Type: [Array](#).

## Example

Query:

```
SELECT groupArrayInsertAt(toString(number), number * 2) FROM numbers(5);
```

Result:

```
groupArrayInsertAt(toString(number), multiply(number, 2))—  
['0','1','2','3','4'] |
```

Query:

```
SELECT groupArrayInsertAt('-')(toString(number), number * 2) FROM numbers(5);
```

Result:

```
groupArrayInsertAt('')(toString(number), multiply(number, 2))—  
['0','-','1','-','2','-','3','-','4'] |
```

Query:

```
SELECT groupArrayInsertAt('-', 5)(toString(number), number * 2) FROM numbers(5);
```

Result:

```
groupArrayInsertAt('-', 5)(toString(number), multiply(number, 2))  
['0','-' '1','-' '2']
```

Multi-threaded insertion of elements into one position.

Query:

```
SELECT groupArrayInsertAt(number, 0) FROM numbers_mt(10) SETTINGS max_block_size = 1;
```

As a result of this query you get random integer in the [0,9] range. For example:

```
groupArrayInsertAt(number, 0)  
[7]
```

## groupArrayMovingSum

Calculates the moving sum of input values.

```
groupArrayMovingSum(numbers_for_summing)  
groupArrayMovingSum(window_size)(numbers_for_summing)
```

The function can take the window size as a parameter. If left unspecified, the function takes the window size equal to the number of rows in the column.

### Arguments

- `numbers_for_summing` — [Expression](#) resulting in a numeric data type value.
- `window_size` — Size of the calculation window.

### Returned values

- Array of the same size and type as the input data.

### Example

The sample table:

```
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

The queries:

```
SELECT
    groupArrayMovingSum(int) AS I,
    groupArrayMovingSum(float) AS F,
    groupArrayMovingSum(dec) AS D
FROM t
```

I	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
```

I	F	D
[1,3,6,11]	[1.1,3.3000002,6.6000004,12.17]	[1.10,3.30,6.60,12.17]

## groupArrayMovingAvg

Calculates the moving average of input values.

```
groupArrayMovingAvg(numbers_for_summing)
groupArrayMovingAvg(window_size)(numbers_for_summing)
```

The function can take the window size as a parameter. If left unspecified, the function takes the window size equal to the number of rows in the column.

### Arguments

- `numbers_for_summing` — **Expression** resulting in a numeric data type value.
- `window_size` — Size of the calculation window.

### Returned values

- Array of the same size and type as the input data.

The function uses **rounding towards zero**. It truncates the decimal places insignificant for the resulting data type.

### Example

The sample table 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

The queries:

```
SELECT
    groupArrayMovingAvg(int) AS I,
    groupArrayMovingAvg(float) AS F,
    groupArrayMovingAvg(dec) AS D
FROM t
```

I	F	D
[0,0,1,3]	[0.275,0.82500005,1.9250001,3.8675]	[0.27,0.82,1.92,3.86]

```
SELECT
    groupArrayMovingAvg(2)(int) AS I,
    groupArrayMovingAvg(2)(float) AS F,
    groupArrayMovingAvg(2)(dec) AS D
FROM t
```

I	F	D
[0,1,3,5]	[0.55,1.6500001,3.3000002,6.085]	[0.55,1.65,3.30,6.08]

## groupArraySample

Creates an array of sample argument values. The size of the resulting array is limited to `max_size` elements. Argument values are selected and added to the array randomly.

### Syntax

```
groupArraySample(max_size[, seed])(x)
```

### Arguments

- `max_size` — Maximum size of the resulting array. `UInt64`.
- `seed` — Seed for the random number generator. Optional. `UInt64`. Default value: `123456`.
- `x` — Argument (column name or expression).

### Returned values

- Array of randomly selected `x` arguments.

Type: **Array**.

## Examples

Consider table colors:

id	color
1	red
2	blue
3	green
4	white
5	orange

Query with column name as argument:

```
SELECT groupArraySample(3)(color) as newcolors FROM colors;
```

Result:

```
newcolors  
['white','blue','green'] |
```

Query with column name and different seed:

```
SELECT groupArraySample(3, 987654321)(color) as newcolors FROM colors;
```

Result:

```
newcolors  
['red','orange','green'] |
```

Query with expression as argument:

```
SELECT groupArraySample(3)(concat('light-', color)) as newcolors FROM colors;
```

Result:

```
newcolors  
['light-blue','light-orange','light-green'] |
```

## groupBitAnd

Applies bitwise AND for series of numbers.

```
groupBitAnd(expr)
```

## Arguments

`expr` – An expression that results in `UInt*` type.

## Return value

Value of the UInt\* type.

## Example

Test data:

```
binary  decimal
00101100 = 44
00011100 = 28
00001101 = 13
01010101 = 85
```

Query:

```
SELECT groupBitAnd(num) FROM t
```

Where num is the column with the test data.

Result:

```
binary  decimal
00000100 = 4
```

# groupBitOr

Applies bitwise OR for series of numbers.

```
groupBitOr(expr)
```

## Arguments

expr – An expression that results in UInt\* type.

## Returned value

Value of the UInt\* type.

## Example

Test data:

```
binary  decimal
00101100 = 44
00011100 = 28
00001101 = 13
01010101 = 85
```

Query:

```
SELECT groupBitOr(num) FROM t
```

Where num is the column with the test data.

Result:

```
binary  decimal  
01111101 = 125
```

## groupBitXor

Applies bitwise XOR for series of numbers.

```
groupBitXor(expr)
```

### Arguments

expr – An expression that results in UInt\* type.

### Return value

Value of the UInt\* type.

### Example

Test data:

```
binary  decimal  
00101100 = 44  
00011100 = 28  
00001101 = 13  
01010101 = 85
```

Query:

```
SELECT groupBitXor(num) FROM t
```

Where num is the column with the test data.

Result:

```
binary  decimal  
01101000 = 104
```

## groupBitmap

Bitmap or Aggregate calculations from a unsigned integer column, return cardinality of type UInt64, if add suffix -State, then return [bitmap object](#).

```
groupBitmap(expr)
```

### Arguments

expr – An expression that results in UInt\* type.

### Return value

Value of the UInt64 type.

### Example

Test data:

```
UserID
1
1
2
3
```

Query:

```
SELECT groupBitmap(UserID) as num FROM t
```

Result:

```
num
3
```

## groupBitmapAnd

Calculations the AND of a bitmap column, return cardinality of type UInt64, if add suffix -State, then return [bitmap object](#).

```
groupBitmapAnd(expr)
```

### Arguments

expr – An expression that results in AggregateFunction(groupBitmap, UInt\*) type.

### Return value

Value of the UInt64 type.

### Example

```
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

Calculations the OR of a bitmap column, return cardinality of type UInt64, if add suffix -State, then return **bitmap object**. This is equivalent to `groupBitmapMerge`.

```
groupBitmapOr(expr)
```

## Arguments

`expr` – An expression that results in `AggregateFunction(groupBitmap, UInt*)` type.

## Returned value

Value of the `UInt64` type.

## Example

```
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

Calculations the XOR of a bitmap column, return cardinality of type UInt64, if add suffix -State, then return **bitmap object**.

```
groupBitmapOr(expr)
```

## Arguments

`expr` – An expression that results in `AggregateFunction(groupBitmap, UInt*)` type.

## Returned value

Value of the `UInt64` type.

## Example

```

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] |

```

## sumWithOverflow

Computes the sum of the numbers, using the same data type for the result as for the input parameters. If the sum exceeds the maximum value for this data type, it is calculated with overflow.

Only works for numbers.

## deltaSum

Sums the arithmetic difference between consecutive rows. If the difference is negative, it is ignored.

### Note

The underlying data must be sorted for this function to work properly. If you would like to use this function in a **materialized view**, you most likely want to use the **deltaSumTimestamp** method instead.

### Syntax

```
deltaSum(value)
```

### Arguments

- `value` — Input values, must be **Integer** or **Float** type.

### Returned value

- A gained arithmetic difference of the **Integer** or **Float** type.

### Examples

Query:

```
SELECT deltaSum(arrayJoin([1, 2, 3]));
```

Result:

```
deltaSum(arrayJoin([1, 2, 3]))  
2 |
```

Query:

```
SELECT deltaSum(arrayJoin([1, 2, 3, 0, 3, 4, 2, 3]));
```

Result:

```
deltaSum(arrayJoin([1, 2, 3, 0, 3, 4, 2, 3]))  
7 |
```

Query:

```
SELECT deltaSum(arrayJoin([2.25, 3, 4.5]));
```

Result:

```
deltaSum(arrayJoin([2.25, 3, 4.5]))  
2.25 |
```

## See Also

- [runningDifference](#)

## deltaSumTimestamp

Adds the difference between consecutive rows. If the difference is negative, it is ignored.

This function is primarily for [materialized views](#) that are ordered by some time bucket-aligned timestamp, for example, a `toStartOfMinute` bucket. Because the rows in such a materialized view will all have the same timestamp, it is impossible for them to be merged in the "right" order. This function keeps track of the timestamp of the values it's seen, so it's possible to order the states correctly during merging.

To calculate the delta sum across an ordered collection you can simply use the [deltaSum](#) function.

### Syntax

```
deltaSumTimestamp(value, timestamp)
```

### Arguments

- `value` — Input values, must be some [Integer](#) type or [Float](#) type or a [Date](#) or [DateTime](#).
- `timestamp` — The parameter for order values, must be some [Integer](#) type or [Float](#) type or a [Date](#) or [DateTime](#).

## Returned value

- Accumulated differences between consecutive values, ordered by the `timestamp` parameter.

Type: `Integer` or `Float` or `Date` or `DateTime`.

## Example

Query:

```
SELECT deltaSumTimestamp(value, timestamp)
FROM (SELECT number AS timestamp, [0, 4, 8, 3, 0, 0, 0, 1, 3, 5][number] AS value FROM numbers(1, 10));
```

Result:

```
deltaSumTimestamp(value, timestamp)---  
13 |-----
```

## sumMap

Syntax: `sumMap(key, value)` or `sumMap(Tuple(key, value))`

Totals the `value` array according to the `key` specified in the `key` array.

Passing tuple of keys and values arrays is a synonym to passing two arrays of keys and values.

The number of elements in `key` and `value` must be the same for each row that is totaled.

Returns a tuple of two arrays: keys in sorted order, and values summed for the corresponding keys.

Example:

```
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]) |
```

## minMap

Syntax: `minMap(key, value)` or `minMap(Tuple(key, value))`

Calculates the minimum from `value` array according to the keys specified in the `key` array.

Passing a tuple of keys and value arrays is identical to passing two arrays of keys and values.

The number of elements in `key` and `value` must be the same for each row that is totaled.

Returns a tuple of two arrays: keys in sorted order, and values calculated for the corresponding keys.

Example:

```
SELECT minMap(a, b)
FROM values('a Array(Int32), b Array(Int64)', ([1, 2], [2, 2]), ([2, 3], [1, 1]))
```

```
minMap(a, b)
([1,2,3],[2,1,1]) |
```

## maxMap

Syntax: `maxMap(key, value)` or `maxMap(Tuple(key, value))`

Calculates the maximum from `value` array according to the keys specified in the `key` array.

Passing a tuple of keys and value arrays is identical to passing two arrays of keys and values.

The number of elements in `key` and `value` must be the same for each row that is totaled.

Returns a tuple of two arrays: keys and values calculated for the corresponding keys.

Example:

```
SELECT maxMap(a, b)
FROM values('a Array(Int32), b Array(Int64)', ([1, 2], [2, 2]), ([2, 3], [1, 1]))
```

```
maxMap(a, b)
([1,2,3],[2,2,1]) |
```

## sumCount

Calculates the sum of the numbers and counts the number of rows at the same time. The function is used by ClickHouse query optimizer: if there are multiple sum, count or avg functions in a query, they can be replaced to single `sumCount` function to reuse the calculations. The function is rarely needed to use explicitly.

### Syntax

```
sumCount(x)
```

### Arguments

- `x` — Input value, must be [Integer](#), [Float](#), or [Decimal](#).

## Returned value

- Tuple (`sum`, `count`), where `sum` is the sum of numbers and `count` is the number of rows with not-NULL values.

Type: [Tuple](#).

## Example

Query:

```
CREATE TABLE s_table (x Int8) Engine = Log;
INSERT INTO s_table SELECT number FROM numbers(0, 20);
INSERT INTO s_table VALUES (NULL);
SELECT sumCount(x) from s_table;
```

Result:

```
sumCount(x)
(190,20)
```

## See also

- [optimize\\_syntax\\_fuse\\_functions](#) setting.

# rankCorr

Computes a rank correlation coefficient.

## Syntax

```
rankCorr(x, y)
```

## Arguments

- `x` — Arbitrary value. [Float32](#) or [Float64](#).
- `y` — Arbitrary value. [Float32](#) or [Float64](#).

## Returned value(s)

- Returns a rank correlation coefficient of the ranks of `x` and `y`. The value of the correlation coefficient ranges from -1 to +1. If less than two arguments are passed, the function will return an exception. The value close to +1 denotes a high linear relationship, and with an increase of one random variable, the second random variable also increases. The value close to -1 denotes a high linear relationship, and with an increase of one random variable, the second random variable decreases. The value close or equal to 0 denotes no relationship between the two random variables.

Type: [Float64](#).

## Example

Query:

```
SELECT rankCorr(number, number) FROM numbers(100);
```

Result:

```
rankCorr(number, number)─  
 1 |
```

Query:

```
SELECT roundBankers(rankCorr(exp(number), sin(number)), 3) FROM numbers(100);
```

Result:

```
roundBankers(rankCorr(exp(number), sin(number)), 3)─  
 -0.037 |
```

## See Also

- [Spearman's rank correlation coefficient](#)

## sumKahan

Calculates the sum of the numbers with [Kahan compensated summation algorithm](#)

Slower than [sum](#) function.

The compensation works only for [Float](#) types.

### Syntax

```
sumKahan(x)
```

### Arguments

- $x$  — Input value, must be [Integer](#), [Float](#), or [Decimal](#).

### Returned value

- the sum of numbers, with type [Integer](#), [Float](#), or [Decimal](#) depends on type of input arguments

### Example

Query:

```
SELECT sum(0.1), sumKahan(0.1) FROM numbers(10);
```

Result:

```
sum(0.1) ─ sumKahan(0.1) ─  
 0.9999999999999999 | 1 |
```

## intervalLengthSum

Calculates the total length of union of all ranges (segments on numeric axis).

## Syntax

```
intervalLengthSum(start, end)
```

## Arguments

- **start** — The starting value of the interval. **Int32**, **Int64**, **UInt32**, **UInt64**, **Float32**, **Float64**, **DateTime** or **Date**.
- **end** — The ending value of the interval. **Int32**, **Int64**, **UInt32**, **UInt64**, **Float32**, **Float64**, **DateTime** or **Date**.

## Note

Arguments must be of the same data type. Otherwise, an exception will be thrown.

## Returned value

- Total length of union of all ranges (segments on numeric axis). Depending on the type of the argument, the return value may be **UInt64** or **Float64** type.

## Examples

1. Input table:

id	start	end
a	1.1	2.9
a	2.5	3.2
a	4	5

In this example, the arguments of the **Float32** type are used. The function returns a value of the **Float64** type.

Result is the sum of lengths of intervals [1.1, 3.2] (union of [1.1, 2.9] and [2.5, 3.2]) and [4, 5]

Query:

```
SELECT id, intervalLengthSum(start, end), toTypeName(intervalLengthSum(start, end)) FROM fl_interval GROUP BY id  
ORDER BY id;
```

Result:

id	intervalLengthSum(start, end)	toTypeName(intervalLengthSum(start, end))
a	3.1	Float64

2. Input table:

id	start	end
a	2020-01-01 01:12:30	2020-01-01 02:10:10
a	2020-01-01 02:05:30	2020-01-01 02:50:31
a	2020-01-01 03:11:22	2020-01-01 03:23:31

In this example, the arguments of the DateTime type are used. The function returns a value in seconds.

Query:

```
SELECT id, intervalLengthSum(start, end), toTypeName(intervalLengthSum(start, end)) FROM dt_interval GROUP BY id  
ORDER BY id;
```

Result:

id	intervalLengthSum(start, end)	toTypeName(intervalLengthSum(start, end))
a	6610   UInt64	

3. Input table:

id	start	end
a	2020-01-01	2020-01-04
a	2020-01-12	2020-01-18

In this example, the arguments of the Date type are used. The function returns a value in days.

Query:

```
SELECT id, intervalLengthSum(start, end), toTypeName(intervalLengthSum(start, end)) FROM date_interval GROUP BY id ORDER BY id;
```

Result:

id	intervalLengthSum(start, end)	toTypeName(intervalLengthSum(start, end))
a	9   UInt64	

## skewPop

Computes the **skewness** of a sequence.

```
skewPop(expr)
```

### Arguments

**expr** — **Expression** returning a number.

### Returned value

The skewness of the given distribution. Type — **Float64**

### Example

```
SELECT skewPop(value) FROM series_with_value_column;
```

## skewSamp

Computes the **sample skewness** of a sequence.

It represents an unbiased estimate of the skewness of a random variable if passed values form its sample.

```
skewSamp(expr)
```

## Arguments

`expr` — **Expression** returning a number.

## Returned value

The skewness of the given distribution. Type — **Float64**. If `n <= 1` (`n` is the size of the sample), then the function returns `nan`.

## Example

```
SELECT skewSamp(value) FROM series_with_value_column;
```

# kurtPop

Computes the **kurtosis** of a sequence.

```
kurtPop(expr)
```

## Arguments

`expr` — **Expression** returning a number.

## Returned value

The kurtosis of the given distribution. Type — **Float64**

## Example

```
SELECT kurtPop(value) FROM series_with_value_column;
```

# kurtSamp

Computes the **sample kurtosis** of a sequence.

It represents an unbiased estimate of the kurtosis of a random variable if passed values form its sample.

```
kurtSamp(expr)
```

## Arguments

`expr` — **Expression** returning a number.

## Returned value

The kurtosis of the given distribution. Type — **Float64**. If `n <= 1` (`n` is a size of the sample), then the function returns `nan`.

## Example

```
SELECT kurtSamp(value) FROM series_with_value_column;
```

# uniq

Calculates the approximate number of different values of the argument.

```
uniq(x[, ...])
```

## Arguments

The function takes a variable number of parameters. Parameters can be `Tuple`, `Array`, `Date`, `DateTime`, `String`, or numeric types.

## Returned value

- A `UInt64`-type number.

## Implementation details

Function:

- Calculates a hash for all parameters in the aggregate, then uses it in calculations.
- Uses an adaptive sampling algorithm. For the calculation state, the function uses a sample of element hash values up to 65536.

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.

- Provides the result deterministically (it does not depend on the query processing order).

We recommend using this function in almost all scenarios.

## See Also

- [uniqCombined](#)
- [uniqCombined64](#)
- [uniqHLL12](#)
- [uniqExact](#)
- [uniqTheta](#)

# uniqExact

Calculates the exact number of different argument values.

```
uniqExact(x[, ...])
```

Use the `uniqExact` function if you absolutely need an exact result. Otherwise use the `uniq` function.

The `uniqExact` function uses more memory than `uniq`, because the size of the state has unbounded growth as the number of different values increases.

## Arguments

The function takes a variable number of parameters. Parameters can be `Tuple`, `Array`, `Date`, `DateTime`, `String`, or numeric types.

## See Also

- [uniq](#)
- [uniqCombined](#)
- [uniqHLL12](#)
- [uniqTheta](#)

# uniqCombined

Calculates the approximate number of different argument values.

```
uniqCombined(HLL_precision)(x[, ...])
```

The `uniqCombined` function is a good choice for calculating the number of different values.

## Arguments

The function takes a variable number of parameters. Parameters can be `Tuple`, `Array`, `Date`, `DateTime`, `String`, or numeric types.

`HLL_precision` is the base-2 logarithm of the number of cells in [HyperLogLog](#). Optional, you can use the function as `uniqCombined(x[, ...])`. The default value for `HLL_precision` is 17, which is effectively 96 KiB of space ( $2^{17}$  cells, 6 bits each).

## Returned value

- A number `UInt64`-type number.

## Implementation details

Function:

- Calculates a hash (64-bit hash for `String` and 32-bit otherwise) for all parameters in the aggregate, then uses it in calculations.
- Uses a combination of three algorithms: array, hash table, and [HyperLogLog](#) with an error correction table.

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.

- Provides the result deterministically (it does not depend on the query processing order).

## Note

Since it uses 32-bit hash for non-String type, the result will have very high error for cardinalities significantly larger than `UINT_MAX` (error will raise quickly after a few tens of billions of distinct values), hence in this case you should use **uniqCombined64**

Compared to the [uniq](#) function, the `uniqCombined`:

- Consumes several times less memory.
- Calculates with several times higher accuracy.
- Usually has slightly lower performance. In some scenarios, `uniqCombined` can perform better than `uniq`, for example, with distributed queries that transmit a large number of aggregation states over the network.

## See Also

- [uniq](#)
- [uniqCombined64](#)
- [uniqHLL12](#)
- [uniqExact](#)
- [uniqTheta](#)

## uniqCombined64

Same as [uniqCombined](#), but uses 64-bit hash for all data types.

## uniqHLL12

Calculates the approximate number of different argument values, using the [HyperLogLog](#) algorithm.

```
uniqHLL12(x[, ...])
```

### Arguments

The function takes a variable number of parameters. Parameters can be `Tuple`, `Array`, `Date`, `DateTime`, `String`, or numeric types.

### Returned value

- A `UInt64`-type number.

### Implementation details

Function:

- Calculates a hash for all parameters in the aggregate, then uses it in calculations.
- Uses the HyperLogLog algorithm to approximate the number of different argument values.

$2^{12}$  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).

- Provides the determinate result (it does not depend on the query processing order).

We do not recommend using this function. In most cases, use the [uniq](#) or [uniqCombined](#) function.

## See Also

- [uniq](#)
- [uniqCombined](#)
- [uniqExact](#)
- [uniqTheta](#)

# uniqTheta

Calculates the approximate number of different argument values, using the [Theta Sketch Framework](#).

```
uniqTheta(x[, ...])
```

## Arguments

The function takes a variable number of parameters. Parameters can be [Tuple](#), [Array](#), [Date](#), [DateTime](#), [String](#), or numeric types.

## Returned value

- A [UInt64](#)-type number.

## Implementation details

Function:

- Calculates a hash for all parameters in the aggregate, then uses it in calculations.
- Uses the [KMV](#) algorithm to approximate the number of different argument values.

4096( $2^{12}$ ) 64-bit sketch are used. The size of the state is about 41 KB.

- The relative error is 3.125% (95% confidence), see the [relative error table](#) for detail.

## See Also

- [uniq](#)
- [uniqCombined](#)
- [uniqCombined64](#)
- [uniqHLL12](#)
- [uniqExact](#)

# quantile

Computes an approximate [quantile](#) of a numeric data sequence.

This function applies [reservoir sampling](#) with a reservoir size up to 8192 and a random number generator for sampling. The result is non-deterministic. To get an exact quantile, use the [quantileExact](#) function.

When using multiple `quantile*` functions with different levels in a query, the internal states are not combined (that is, the query works less efficiently than it could). In this case, use the [quantiles](#) function.

## Syntax

```
quantile(level)(expr)
```

Alias: `median`.

## Arguments

- `level` — Level of quantile. Optional parameter. Constant floating-point number from 0 to 1. We recommend using a `level` value in the range of [0.01, 0.99]. Default value: 0.5. At `level=0.5` the function calculates [median](#).
- `expr` — Expression over the column values resulting in numeric [data types](#), [Date](#) or [DateTime](#).

## Returned value

- Approximate quantile of the specified level.

Type:

- [Float64](#) for numeric data type input.
- [Date](#) if input values have the `Date` type.
- [DateTime](#) if input values have the `DateTime` type.

## Example

Input table:

val
1
1
2
3

Query:

```
SELECT quantile(val) FROM t
```

Result:

quantile(val)
1.5

## See Also

- [median](#)
- [quantiles](#)

# quantiles Functions

## quantiles

Syntax: `quantiles(level1, level2, ...)(x)`

All the quantile functions also have corresponding quantiles functions: `quantiles`, `quantilesDeterministic`, `quantilesTiming`, `quantilesTimingWeighted`, `quantilesExact`, `quantilesExactWeighted`, `quantilesTDigest`, `quantilesBFloat16`. These functions calculate all the quantiles of the listed levels in one pass, and return an array of the resulting values.

## quantilesExactExclusive

Exactly computes the `quantiles` of a numeric data sequence.

To get exact value, all the passed values are combined into an array, which is then partially sorted. Therefore, the function consumes  $O(n)$  memory, where  $n$  is a number of values that were passed. However, for a small number of values, the function is very effective.

This function is equivalent to `PERCENTILE.EXC` Excel function, ([type R6](#)).

Works more efficiently with sets of levels than `quantileExactExclusive`.

### Syntax

```
quantilesExactExclusive(level1, level2, ...)(expr)
```

### Arguments

- `expr` — Expression over the column values resulting in numeric [data types](#), [Date](#) or [DateTime](#).

### Parameters

- `level` — Levels of quantiles. Possible values: (0, 1) — bounds not included. [Float](#).

### Returned value

- [Array](#) of quantiles of the specified levels.

Type of array values:

- [Float64](#) for numeric data type input.
- [Date](#) if input values have the [Date](#) type.
- [DateTime](#) if input values have the [DateTime](#) type.

### Example

Query:

```
CREATE TABLE num AS numbers(1000);
SELECT quantilesExactExclusive(0.25, 0.5, 0.75, 0.9, 0.95, 0.99, 0.999)(x) FROM (SELECT number AS x FROM num);
```

Result:

```
quantilesExactExclusive(0.25, 0.5, 0.75, 0.9, 0.95, 0.99, 0.999)(x) └
```

```
[249.25,499.5,749.75,899.9,949.949999999999,989.99,998.999]
```

```
|
```

```
└
```

## quantilesExactInclusive

Exactly computes the [quantiles](#) of a numeric data sequence.

To get exact value, all the passed values are combined into an array, which is then partially sorted.

Therefore, the function consumes  $O(n)$  memory, where  $n$  is a number of values that were passed. However, for a small number of values, the function is very effective.

This function is equivalent to [PERCENTILE.INC](#) Excel function, ([type R7](#)).

Works more efficiently with sets of levels than [quantileExactInclusive](#).

### Syntax

```
quantilesExactInclusive(level1, level2, ...)(expr)
```

### Arguments

- `expr` — Expression over the column values resulting in numeric [data types](#), [Date](#) or [DateTime](#).

### Parameters

- `level` — Levels of quantiles. Possible values: [0, 1] — bounds included. [Float](#).

### Returned value

- [Array](#) of quantiles of the specified levels.

Type of array values:

- [Float64](#) for numeric data type input.
- [Date](#) if input values have the [Date](#) type.
- [DateTime](#) if input values have the [DateTime](#) type.

### Example

Query:

```
CREATE TABLE num AS numbers(1000);
```

```
SELECT quantilesExactInclusive(0.25, 0.5, 0.75, 0.9, 0.95, 0.99, 0.999)(x) FROM (SELECT number AS x FROM num);
```

Result:

```
quantilesExactInclusive(0.25, 0.5, 0.75, 0.9, 0.95, 0.99, 0.999)(x) └
```

```
[249.75,499.5,749.25,899.1,949.05,989.01,998.001]
```

```
|
```

```
└
```

## quantileExact Functions

### quantileExact

Exactly computes the [quantile](#) of a numeric data sequence.

To get exact value, all the passed values are combined into an array, which is then partially sorted.

Therefore, the function consumes  $O(n)$  memory, where  $n$  is a number of values that were passed. However, for a small number of values, the function is very effective.

When using multiple `quantile*` functions with different levels in a query, the internal states are not combined (that is, the query works less efficiently than it could). In this case, use the [quantiles](#) function.

## Syntax

```
quantileExact(level)(expr)
```

Alias: `medianExact`.

## Arguments

- `level` — Level of quantile. Optional parameter. Constant floating-point number from 0 to 1. We recommend using a `level` value in the range of `[0.01, 0.99]`. Default value: 0.5. At `level=0.5` the function calculates [median](#).
- `expr` — Expression over the column values resulting in numeric [data types](#), [Date](#) or [DateTime](#).

## Returned value

- Quantile of the specified level.

Type:

- [Float64](#) for numeric data type input.
- [Date](#) if input values have the `Date` type.
- [DateTime](#) if input values have the `DateTime` type.

## Example

Query:

```
SELECT quantileExact(number) FROM numbers(10)
```

Result:

```
quantileExact(number)─  
5 |
```

## quantileExactLow

Similar to `quantileExact`, this computes the exact [quantile](#) of a numeric data sequence.

To get the exact value, all the passed values are combined into an array, which is then fully sorted. The sorting [algorithm's](#) complexity is  $O(N \cdot \log(N))$ , where  $N = \text{std}::\text{distance}(\text{first}, \text{last})$  comparisons.

The return value depends on the quantile level and the number of elements in the selection, i.e. if the level is 0.5, then the function returns the lower median value for an even number of elements and the middle median value for an odd number of elements. Median is calculated similarly to the [median\\_low](#) implementation which is used in python.

For all other levels, the element at the index corresponding to the value of `level * size_of_array` is returned.  
For example:

```
SELECT quantileExactLow(0.1)(number) FROM numbers(10)
```

```
└─quantileExactLow(0.1)(number)─
```

```
  1 |
```

When using multiple `quantile*` functions with different levels in a query, the internal states are not combined (that is, the query works less efficiently than it could). In this case, use the `quantiles` function.

## Syntax

```
quantileExactLow(level)(expr)
```

Alias: `medianExactLow`.

## Arguments

- `level` — Level of quantile. Optional parameter. Constant floating-point number from 0 to 1. We recommend using a `level` value in the range of [0.01, 0.99]. Default value: 0.5. At `level=0.5` the function calculates `median`.
- `expr` — Expression over the column values resulting in numeric `data types`, `Date` or `DateTime`.

## Returned value

- Quantile of the specified level.

Type:

- `Float64` for numeric data type input.
- `Date` if input values have the `Date` type.
- `DateTime` if input values have the `DateTime` type.

## Example

Query:

```
SELECT quantileExactLow(number) FROM numbers(10)
```

Result:

```
└─quantileExactLow(number)─
```

```
  4 |
```

## quantileExactHigh

Similar to `quantileExact`, this computes the exact `quantile` of a numeric data sequence.

All the passed values are combined into an array, which is then fully sorted, to get the exact value. The sorting `algorithm's` complexity is  $O(N \cdot \log(N))$ , where `N = std::distance(first, last)` comparisons.

The return value depends on the quantile level and the number of elements in the selection, i.e. if the level is 0.5, then the function returns the higher median value for an even number of elements and the middle median value for an odd number of elements. Median is calculated similarly to the [median\\_high](#) implementation which is used in python. For all other levels, the element at the index corresponding to the value of `level * size_of_array` is returned.

This implementation behaves exactly similar to the current `quantileExact` implementation.

When using multiple `quantile*` functions with different levels in a query, the internal states are not combined (that is, the query works less efficiently than it could). In this case, use the [quantiles](#) function.

## Syntax

```
quantileExactHigh(level)(expr)
```

Alias: `medianExactHigh`.

## Arguments

- `level` — Level of quantile. Optional parameter. Constant floating-point number from 0 to 1. We recommend using a `level` value in the range of `[0.01, 0.99]`. Default value: 0.5. At `level=0.5` the function calculates [median](#).
- `expr` — Expression over the column values resulting in numeric [data types](#), [Date](#) or [DateTime](#).

## Returned value

- Quantile of the specified level.

Type:

- [Float64](#) for numeric data type input.
- [Date](#) if input values have the `Date` type.
- [DateTime](#) if input values have the `DateTime` type.

## Example

Query:

```
SELECT quantileExactHigh(number) FROM numbers(10)
```

Result:

```
quantileExactHigh(number)─  
      5 |
```

# quantileExactExclusive

Exactly computes the [quantile](#) of a numeric data sequence.

To get exact value, all the passed values are combined into an array, which is then partially sorted. Therefore, the function consumes  $O(n)$  memory, where  $n$  is a number of values that were passed. However, for a small number of values, the function is very effective.

This function is equivalent to [PERCENTILE.EXC](#) Excel function, ([type R6](#)).

When using multiple `quantileExactExclusive` functions with different levels in a query, the internal states are not combined (that is, the query works less efficiently than it could). In this case, use the `quantilesExactExclusive` function.

## Syntax

```
quantileExactExclusive(level)(expr)
```

## Arguments

- `expr` — Expression over the column values resulting in numeric [data types](#), [Date](#) or [DateTime](#).

## Parameters

- `level` — Level of quantile. Optional. Possible values: (0, 1) — bounds not included. Default value: 0.5. At `level=0.5` the function calculates [median](#). [Float](#).

## Returned value

- Quantile of the specified level.

Type:

- [Float64](#) for numeric data type input.
- [Date](#) if input values have the `Date` type.
- [DateTime](#) if input values have the `DateTime` type.

## Example

Query:

```
CREATE TABLE num AS numbers(1000);
SELECT quantileExactExclusive(0.6)(x) FROM (SELECT number AS x FROM num);
```

Result:

```
quantileExactExclusive(0.6)(x)─
 599.6 |
```

## quantileExactInclusive

Exactly computes the [quantile](#) of a numeric data sequence.

To get exact value, all the passed values are combined into an array, which is then partially sorted. Therefore, the function consumes  $O(n)$  memory, where  $n$  is a number of values that were passed. However, for a small number of values, the function is very effective.

This function is equivalent to [PERCENTILE.INC](#) Excel function, ([type R7](#)).

When using multiple `quantileExactInclusive` functions with different levels in a query, the internal states are not combined (that is, the query works less efficiently than it could). In this case, use the `quantilesExactInclusive` function.

## Syntax

```
quantileExactInclusive(level)(expr)
```

## Arguments

- `expr` — Expression over the column values resulting in numeric [data types](#), [Date](#) or [DateTime](#).

## Parameters

- `level` — Level of quantile. Optional. Possible values: [0, 1] — bounds included. Default value: 0.5. At `level=0.5` the function calculates [median](#). [Float](#).

## Returned value

- Quantile of the specified level.

Type:

- [Float64](#) for numeric data type input.
- [Date](#) if input values have the [Date](#) type.
- [DateTime](#) if input values have the [DateTime](#) type.

## Example

Query:

```
CREATE TABLE num AS numbers(1000);
SELECT quantileExactInclusive(0.6)(x) FROM (SELECT number AS x FROM num);
```

Result:

```
quantileExactInclusive(0.6)(x)
599.4 |
```

## See Also

- [median](#)
- [quantiles](#)

# quantileExactWeighted

Exactly computes the [quantile](#) of a numeric data sequence, taking into account the weight of each element.

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](#). You can use this function instead of [quantileExact](#) and specify the weight 1.

When using multiple `quantile*` functions with different levels in a query, the internal states are not combined (that is, the query works less efficiently than it could). In this case, use the [quantiles](#) function.

## Syntax

```
quantileExactWeighted(level)(expr, weight)
```

Alias: `medianExactWeighted`.

## Arguments

- `level` — Level of quantile. Optional parameter. Constant floating-point number from 0 to 1. We recommend using a `level` value in the range of [0.01, 0.99]. Default value: 0.5. At `level=0.5` the function calculates `median`.
- `expr` — Expression over the column values resulting in numeric [data types](#), [Date](#) or [DateTime](#).
- `weight` — Column with weights of sequence members. Weight is a number of value occurrences.

## Returned value

- Quantile of the specified level.

Type:

- [Float64](#) for numeric data type input.
- [Date](#) if input values have the `Date` type.
- [DateTime](#) if input values have the `DateTime` type.

## Example

Input table:

n	val
0	3
1	2
2	1
5	4

Query:

```
SELECT quantileExactWeighted(n, val) FROM t
```

Result:

```
quantileExactWeighted(n, val)
  1 |
```

## See Also

- [median](#)
- [quantiles](#)

# quantileTiming

With the determined precision computes the [quantile](#) of a numeric data sequence.

The result is deterministic (it does not depend on the query processing order). The function is optimized for working with sequences which describe distributions like loading web pages times or backend response times.

When using multiple `quantile*` functions with different levels in a query, the internal states are not combined (that is, the query works less efficiently than it could). In this case, use the `quantiles` function.

## Syntax

```
quantileTiming(level)(expr)
```

Alias: `medianTiming`.

## Arguments

- `level` — Level of quantile. Optional parameter. Constant floating-point number from 0 to 1. We recommend using a `level` value in the range of `[0.01, 0.99]`. Default value: 0.5. At `level=0.5` the function calculates `median`.
- `expr` — **Expression** over a column values returning a `Float*`-type number.
  - 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.

## Accuracy

The calculation is accurate if:

- Total number of values does not exceed 5670.
- Total number of values exceeds 5670, but the page loading time is less than 1024ms.

Otherwise, the result of the calculation is rounded to the nearest multiple of 16 ms.

## Note

For calculating page loading time quantiles, this function is more effective and accurate than `quantile`.

## Returned value

- Quantile of the specified level.

Type: `Float32`.

## Note

If no values are passed to the function (when using `quantileTimingIf`), `NaN` is returned. The purpose of this is to differentiate these cases from cases that result in zero. See **ORDER BY clause** for notes on sorting `NaN` values.

## Example

Input table:

```
└── response_time ──  
    72 |  
    112 |  
    126 |  
    145 |  
    104 |  
    242 |  
    313 |  
    168 |  
    108 |
```

Query:

```
SELECT quantileTiming(response_time) FROM t
```

Result:

```
└── quantileTiming(response_time) ──  
    126 |
```

## See Also

- [median](#)
- [quantiles](#)

# quantileTimingWeighted

With the determined precision computes the [quantile](#) of a numeric data sequence according to the weight of each sequence member.

The result is deterministic (it does not depend on the query processing order). The function is optimized for working with sequences which describe distributions like loading web pages times or backend response times.

When using multiple `quantile*` functions with different levels in a query, the internal states are not combined (that is, the query works less efficiently than it could). In this case, use the [quantiles](#) function.

## Syntax

```
quantileTimingWeighted(level)(expr, weight)
```

Alias: `medianTimingWeighted`.

## Arguments

- **level** — Level of quantile. Optional parameter. Constant floating-point number from 0 to 1. We recommend using a `level` value in the range of [0.01, 0.99]. Default value: 0.5. At `level=0.5` the function calculates [median](#).
- **expr** — [Expression](#) over a column values returning a [Float\\*](#)-type number.

- 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.

## Accuracy

The calculation is accurate if:

- Total number of values does not exceed 5670.
- Total number of values exceeds 5670, but the page loading time is less than 1024ms.

Otherwise, the result of the calculation is rounded to the nearest multiple of 16 ms.

## Note

For calculating page loading time quantiles, this function is more effective and accurate than [quantile](#).

## Returned value

- Quantile of the specified level.

Type: `Float32`.

## Note

If no values are passed to the function (when using `quantileTimingIf`), **NaN** is returned. The purpose of this is to differentiate these cases from cases that result in zero. See [ORDER BY clause](#) for notes on sorting `NaN` values.

## Example

Input table:

response_time	weight
68	1
104	2
112	3
126	2
138	1
162	1

Query:

```
SELECT quantileTimingWeighted(response_time, weight) FROM t
```

Result:

```
quantileTimingWeighted(response_time, weight)
112
```

# quantilesTimingWeighted

Same as `quantileTimingWeighted`, but accept multiple parameters with quantile levels and return an Array filled with many values of that quantiles.

## Example

Input table:

response_time	weight
68	1
104	2
112	3
126	2
138	1
162	1

Query:

```
SELECT quantilesTimingWeighted(0.5, 0.99)(response_time, weight) FROM t
```

Result:

```
quantilesTimingWeighted(0.5, 0.99)(response_time, weight)
[112,162]
```

## See Also

- [median](#)
- [quantiles](#)

# quantileDeterministic

Computes an approximate [quantile](#) of a numeric data sequence.

This function applies [reservoir sampling](#) with a reservoir size up to 8192 and deterministic algorithm of sampling. The result is deterministic. To get an exact quantile, use the [quantileExact](#) function.

When using multiple `quantile*` functions with different levels in a query, the internal states are not combined (that is, the query works less efficiently than it could). In this case, use the [quantiles](#) function.

## Syntax

```
quantileDeterministic(level)(expr, determinator)
```

Alias: `medianDeterministic`.

## Arguments

- `level` — Level of quantile. Optional parameter. Constant floating-point number from 0 to 1. We recommend using a `level` value in the range of `[0.01, 0.99]`. Default value: 0.5. At `level=0.5` the function calculates [median](#).
- `expr` — Expression over the column values resulting in numeric [data types](#), [Date](#) or [DateTime](#).

- **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.

## Returned value

- Approximate quantile of the specified level.

Type:

- **Float64** for numeric data type input.
- **Date** if input values have the **Date** type.
- **DateTime** if input values have the **DateTime** type.

## Example

Input table:

val
1
1
2
3

Query:

```
SELECT quantileDeterministic(val, 1) FROM t
```

Result:

```
quantileDeterministic(val, 1)─  
1.5 |
```

## See Also

- [median](#)
- [quantiles](#)

# quantileTDigest

Computes an approximate **quantile** of a numeric data sequence using the **t-digest** algorithm.

Memory consumption is  $\log(n)$ , where  $n$  is a number of values. The result depends on the order of running the query, and is nondeterministic.

The performance of the function is lower than performance of **quantile** or **quantileTiming**. In terms of the ratio of State size to precision, this function is much better than **quantile**.

When using multiple **quantile\*** functions with different levels in a query, the internal states are not combined (that is, the query works less efficiently than it could). In this case, use the **quantiles** function.

## Syntax

```
quantileTDigest(level)(expr)
```

Alias: `medianTDigest`.

## Arguments

- `level` — Level of quantile. Optional parameter. Constant floating-point number from 0 to 1. We recommend using a `level` value in the range of [0.01, 0.99]. Default value: 0.5. At `level=0.5` the function calculates `median`.
- `expr` — Expression over the column values resulting in numeric [data types](#), [Date](#) or [DateTime](#).

## Returned value

- Approximate quantile of the specified level.

Type:

- [Float64](#) for numeric data type input.
- [Date](#) if input values have the `Date` type.
- [DateTime](#) if input values have the `DateTime` type.

## Example

Query:

```
SELECT quantileTDigest(number) FROM numbers(10)
```

Result:

```
quantileTDigest(number)─  
4.5 |
```

## See Also

- [median](#)
- [quantiles](#)

# quantileTDigestWeighted

Computes an approximate [quantile](#) of a numeric data sequence using the [t-digest](#) algorithm. The function takes into account the weight of each sequence member. The maximum error is 1%. Memory consumption is  $\log(n)$ , where `n` is a number of values.

The performance of the function is lower than performance of [quantile](#) or [quantileTiming](#). In terms of the ratio of State size to precision, this function is much better than [quantile](#).

The result depends on the order of running the query, and is nondeterministic.

When using multiple `quantile*` functions with different levels in a query, the internal states are not combined (that is, the query works less efficiently than it could). In this case, use the [quantiles](#) function.

## Note

Using `quantileTDigestWeighted` is not recommended for tiny data sets and can lead to significant error. In this case, consider possibility of using `quantileTDigest` instead.

## Syntax

```
quantileTDigestWeighted(level)(expr, weight)
```

Alias: `medianTDigestWeighted`.

## Arguments

- `level` — Level of quantile. Optional parameter. Constant floating-point number from 0 to 1. We recommend using a `level` value in the range of [0.01, 0.99]. Default value: 0.5. At `level=0.5` the function calculates `median`.
- `expr` — Expression over the column values resulting in numeric [data types](#), [Date](#) or [DateTime](#).
- `weight` — Column with weights of sequence elements. Weight is a number of value occurrences.

## Returned value

- Approximate quantile of the specified level.

Type:

- [Float64](#) for numeric data type input.
- [Date](#) if input values have the `Date` type.
- [DateTime](#) if input values have the `DateTime` type.

## Example

Query:

```
SELECT quantileTDigestWeighted(number, 1) FROM numbers(10)
```

Result:

```
quantileTDigestWeighted(number, 1)─  
4.5 |
```

## See Also

- [median](#)
- [quantiles](#)

# quantileBFloat16

Computes an approximate `quantile` of a sample consisting of `bfloat16` numbers. `bfloat16` is a floating-point data type with 1 sign bit, 8 exponent bits and 7 fraction bits.

The function converts input values to 32-bit floats and takes the most significant 16 bits. Then it calculates `bfloat16` quantile value and converts the result to a 64-bit float by appending zero bits.

The function is a fast quantile estimator with a relative error no more than 0.390625%.

## Syntax

```
quantileBFloat16[(level)](#sql-reference-aggregate-functions-reference-expr)
```

Alias: medianBFloat16

## Arguments

- `expr` — Column with numeric data. [Integer](#), [Float](#).

## Parameters

- `level` — Level of quantile. Optional. Possible values are in the range from 0 to 1. Default value: 0.5. [Float](#).

## Returned value

- Approximate quantile of the specified level.

Type: [Float64](#).

## Example

Input table has an integer and a float columns:

a	b
1	1.001
2	1.002
3	1.003
4	1.004

Query to calculate 0.75-quantile (third quartile):

```
SELECT quantileBFloat16(0.75)(a), quantileBFloat16(0.75)(b) FROM example_table;
```

Result:

```
quantileBFloat16(0.75)(a)  quantileBFloat16(0.75)(b)
 3 |          1 |
```

Note that all floating point values in the example are truncated to 1.0 when converting to `bfloat16`.

## quantileBFloat16Weighted

Like `quantileBFloat16` but takes into account the weight of each sequence member.

## See Also

- [median](#)
- [quantiles](#)

## median

The `median*` functions are the aliases for the corresponding `quantile*` functions. They calculate median of a numeric data sample.

Functions:

- `median` — Alias for `quantile`.
- `medianDeterministic` — Alias for `quantileDeterministic`.
- `medianExact` — Alias for `quantileExact`.
- `medianExactWeighted` — Alias for `quantileExactWeighted`.
- `medianTiming` — Alias for `quantileTiming`.
- `medianTimingWeighted` — Alias for `quantileTimingWeighted`.
- `medianTDigest` — Alias for `quantileTDigest`.
- `medianTDigestWeighted` — Alias for `quantileTDigestWeighted`.
- `medianBFloat16` — Alias for `quantileBFloat16`.

## Example

Input table:

val
1
1
2
3

Query:

```
SELECT medianDeterministic(val, 1) FROM t;
```

Result:

```
medianDeterministic(val, 1)  
1.5 |
```

## simpleLinearRegression

Performs simple (unidimensional) linear regression.

```
simpleLinearRegression(x, y)
```

Parameters:

- `x` — Column with dependent variable values.
- `y` — Column with explanatory variable values.

Returned values:

Constants (`a`, `b`) of the resulting line  $y = a*x + b$ .

## Examples

```
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)
```

## stochasticLinearRegression

This function implements stochastic linear regression. It supports custom parameters for learning rate, L2 regularization coefficient, mini-batch size and has few methods for updating weights ([Adam](#) (used by default), [simple SGD](#), [Momentum](#), [Nesterov](#)).

### Parameters

There are 4 customizable parameters. They are passed to the function sequentially, but there is no need to pass all four - default values will be used, however good model required some parameter tuning.

```
stochasticLinearRegression(1.0, 1.0, 10, 'SGD')
```

1. learning rate is the coefficient on step length, when gradient descent step is performed. Too big learning rate may cause infinite weights of the model. Default is 0.00001.
2. L2 regularization coefficient which may help to prevent overfitting. Default is 0.1.
3. mini-batch size sets the number of elements, which gradients will be computed and summed to perform one step of gradient descent. Pure stochastic descent uses one element, however having small batches (about 10 elements) make gradient steps more stable. Default is 15.
4. method for updating weights, they are: Adam (by default), SGD, Momentum, Nesterov. Momentum and Nesterov require little bit more computations and memory, however they happen to be useful in terms of speed of convergance and stability of stochastic gradient methods.

### Usage

`stochasticLinearRegression` is used in two steps: fitting the model and predicting on new data. In order to fit the model and save its state for later usage we use `-State` combinator, which basically saves the state (model weights, etc).

To predict we use function [evalMLMethod](#), which takes a state as an argument as well as features to predict on.

#### 1. Fitting

Such query may be used.

```

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;

```

Here we also need to insert data into `train_data` table. The number of parameters is not fixed, it depends only on number of arguments, passed into `linearRegressionState`. They all must be numeric values. Note that the column with target value(which we would like to learn to predict) is inserted as the first argument.

## 2. Predicting

After saving a state into the table, we may use it multiple times for prediction, or even merge with other states and create new even better models.

```

WITH (SELECT state FROM your_model) AS model SELECT
evalMLMethod(model, param1, param2) FROM test_data

```

The query will return a column of predicted values. Note that first argument of `evalMLMethod` is `AggregateFunctionState` object, next are columns of features.

`test_data` is a table like `train_data` but may not contain target value.

## Notes

1. To merge two models user may create such query:

```
sql SELECT state1 + state2 FROM your_models
```

where `your_models` table contains both models. This query will return new `AggregateFunctionState` object.

2. User may fetch weights of the created model for its own purposes without saving the model if no-State combinator is used.

```
sql SELECT stochasticLinearRegression(0.01)(target, param1, param2) FROM train_data
```

Such query will fit the model and return its weights - first are weights, which correspond to the parameters of the model, the last one is bias. So in the example above the query will return a column with 3 values.

## See Also

- [stochasticLogisticRegression](#)
- [Difference between linear and logistic regressions](#)

## stochasticLogisticRegression

This function implements stochastic logistic regression. It can be used for binary classification problem, supports the same custom parameters as `stochasticLinearRegression` and works the same way.

## Parameters

Parameters are exactly the same as in `stochasticLinearRegression`: learning rate, l2 regularization coefficient, mini-batch size, method for updating weights. For more information see [parameters](#).

```
stochasticLogisticRegression(1.0, 1.0, 10, 'SGD')
```

## 1. Fitting

See the `Fitting` section in the [stochasticLinearRegression](#stochasticlinearregression-usage-fitting) description.

Predicted labels have to be in  $[-1, 1]$ .

## 2. Predicting

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.

## See Also

- [stochasticLinearRegression](#)
- [Difference between linear and logistic regressions.](#)

## categoricalInformationValue

Calculates the value of  $(P(\text{tag} = 1) - P(\text{tag} = 0))(\log(P(\text{tag} = 1)) - \log(P(\text{tag} = 0)))$  for each category.

```
categoricalInformationValue(category1, category2, ..., tag)
```

The result indicates how a discrete (categorical) feature [category1, category2, ...] contribute to a learning model which predicting the value of tag.

## studentTTest

Applies Student's t-test to samples from two populations.

### Syntax

```
studentTTest(sample_data, sample_index)
```

Values of both samples are in the `sample_data` column. If `sample_index` equals to 0 then the value in that row belongs to the sample from the first population. Otherwise it belongs to the sample from the second population.

The null hypothesis is that means of populations are equal. Normal distribution with equal variances is assumed.

## Arguments

- `sample_data` — Sample data. [Integer](#), [Float](#) or [Decimal](#).
- `sample_index` — Sample index. [Integer](#).

## Returned values

[Tuple](#) with two elements:

- calculated t-statistic. [Float64](#).
- calculated p-value. [Float64](#).

## Example

Input table:

| sample_data | sample_index |
|-------------|--------------|
| 20.3        | 0            |
| 21.1        | 0            |
| 21.9        | 1            |
| 21.7        | 0            |
| 19.9        | 1            |
| 21.8        | 1            |

Query:

```
SELECT studentTTest(sample_data, sample_index) FROM student_ttest;
```

Result:

```
studentTTest(sample_data, sample_index)
(-0.21739130434783777,0.8385421208415731) |
```

## See Also

- [Student's t-test](#)
- [welchTTest function](#)

# welchTTest

Applies Welch's t-test to samples from two populations.

## Syntax

```
welchTTest(sample_data, sample_index)
```

Values of both samples are in the `sample_data` column. If `sample_index` equals to 0 then the value in that row belongs to the sample from the first population. Otherwise it belongs to the sample from the second population.

The null hypothesis is that means of populations are equal. Normal distribution is assumed. Populations may have unequal variance.

## Arguments

- `sample_data` — Sample data. [Integer](#), [Float](#) or [Decimal](#).
- `sample_index` — Sample index. [Integer](#).

## Returned values

[Tuple](#) with two elements:

- calculated t-statistic. [Float64](#).
- calculated p-value. [Float64](#).

## Example

Input table:

| sample_data | sample_index |
|-------------|--------------|
| 20.3        | 0            |
| 22.1        | 0            |
| 21.9        | 0            |
| 18.9        | 1            |
| 20.3        | 1            |
| 19          | 1            |

Query:

```
SELECT welchTTest(sample_data, sample_index) FROM welch_ttest;
```

Result:

```
welchTTest(sample_data, sample_index)
(2.7988719532211235,0.051807360348581945) |
```

## See Also

- [Welch's t-test](#)
- [studentTTest function](#)

# entropy

Calculates [Shannon entropy](#) of a column of values.

## Syntax

```
entropy(val)
```

## Arguments

- `val` — Column of values of any type.

## Returned value

- Shannon entropy.

Type: `Float64`.

## Example

Query:

```
CREATE TABLE entropy(`vals` UInt32, `strings` String) ENGINE = Memory;
INSERT INTO entropy VALUES (1, 'A'), (1, 'A'), (1, 'A'), (1, 'A'), (2, 'B'), (2, 'B'), (2, 'C'), (2, 'D');
SELECT entropy(vals), entropy(strings) FROM entropy;
```

Result:

| entropy(vals) | entropy(strings) |
|---------------|------------------|
| 1             | 1.75             |

# mannWhitneyUTest

Applies the Mann-Whitney rank test to samples from two populations.

## Syntax

```
mannWhitneyUTest[(alternative[, continuity_correction])](sample_data, sample_index)
```

Values of both samples are in the `sample_data` column. If `sample_index` equals to 0 then the value in that row belongs to the sample from the first population. Otherwise it belongs to the sample from the second population.

The null hypothesis is that two populations are stochastically equal. Also one-sided hypotheses can be tested. This test does not assume that data have normal distribution.

## Arguments

- `sample_data` — sample data. `Integer`, `Float` or `Decimal`.
- `sample_index` — sample index. `Integer`.

## Parameters

- `alternative` — alternative hypothesis. (Optional, default: `'two-sided'`.) `String`.
  - `'two-sided'`;
  - `'greater'`;
  - `'less'`.
- `continuity_correction` — if not 0 then continuity correction in the normal approximation for the p-value is applied. (Optional, default: 1.) `UInt64`.

## Returned values

`Tuple` with two elements:

- calculated U-statistic. [Float64](#).
- calculated p-value. [Float64](#).

## Example

Input table:

| sample_data | sample_index |
|-------------|--------------|
| 10          | 0            |
| 11          | 0            |
| 12          | 0            |
| 1           | 1            |
| 2           | 1            |
| 3           | 1            |

Query:

```
SELECT mannWhitneyUTest('greater')(sample_data, sample_index) FROM mww_ttest;
```

Result:

|                                                        |  |
|--------------------------------------------------------|--|
| mannWhitneyUTest('greater')(sample_data, sample_index) |  |
| (9.0, 0.04042779918503192)                             |  |

## See Also

- [Mann-Whitney U test](#)
- [Stochastic ordering](#)

# Aggregate Function Combinators

The name of an aggregate function can have a suffix appended to it. This changes the way the aggregate function works.

## -If

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).

Examples: `sumIf(column, cond)`, `countIf(cond)`, `avgIf(x, cond)`, `quantilesTimingIf(level1, level2)(x, cond)`, `argMinIf(arg, val, cond)` and so on.

With conditional aggregate functions, you can calculate aggregates for several conditions at once, without using subqueries and JOINs. For example, in Yandex.Metrica, conditional aggregate functions are used to implement the segment comparison functionality.

## -Array

The -Array suffix can be appended to any aggregate function. In this case, the aggregate function takes arguments of the 'Array(T)' type (arrays) instead of 'T' type arguments. If the aggregate function accepts multiple arguments, this must be arrays of equal lengths. When processing arrays, the aggregate function works like the original aggregate function across all array elements.

Example 1: `sumArray(arr)` - Totals all the elements of all ‘arr’ arrays. In this example, it could have been written more simply: `sum(arraySum(arr))`.

Example 2: `uniqArray(arr)` – Counts the number of unique elements in all ‘arr’ arrays. This could be done an easier way: `uniq(arrayJoin(arr))`, but it’s not always possible to add ‘arrayJoin’ to a query.

-If and -Array can be combined. However, ‘Array’ must come first, then ‘If’. Examples: `uniqArrayIf(arr, cond)`, `quantilesTimingArrayIf(level1, level2)(arr, cond)`. Due to this order, the ‘cond’ argument won’t be an array.

## -SimpleState

If you apply this combinator, the aggregate function returns the same value but with a different type. This is a [SimpleAggregateFunction\(...\)](#) that can be stored in a table to work with [AggregatingMergeTree](#) tables.

### Syntax

```
<aggFunction>SimpleState(x)
```

### Arguments

- `x` — Aggregate function parameters.

### Returned values

The value of an aggregate function with the [SimpleAggregateFunction\(...\)](#) type.

### Example

Query:

```
WITH anySimpleState(number) AS c SELECT toTypeName(c), c FROM numbers(1);
```

Result:

```
toTypeName(c)
SimpleAggregateFunction(any, UInt64) | 0 |c
```

## -State

If you apply this combinator, the aggregate function does not return the resulting value (such as the number of unique values for the `uniq` function), but an intermediate state of the aggregation (for `uniq`, this is the hash table for calculating the number of unique values). This is an [AggregateFunction\(...\)](#) that can be used for further processing or stored in a table to finish aggregating later.

To work with these states, use:

- [AggregatingMergeTree](#) table engine.
- [finalizeAggregation](#) function.
- [runningAccumulate](#) function.
- [-Merge](#) combinator.
- [-MergeState](#) combinator.

## -Merge

If you apply this combinator, the aggregate function takes the intermediate aggregation state as an argument, combines the states to finish aggregation, and returns the resulting value.

## -MergeState

Merges the intermediate aggregation states in the same way as the -Merge combinator. However, it does not return the resulting value, but an intermediate aggregation state, similar to the -State combinator.

## -ForEach

Converts an aggregate function for tables into an aggregate function for arrays that aggregates the corresponding array items and returns an array of results. For example, `sumForEach` for the arrays [1, 2], [3, 4, 5] and [6, 7] returns the result [10, 13, 5] after adding together the corresponding array items.

## -Distinct

Every unique combination of arguments will be aggregated only once. Repeating values are ignored.

Examples: `sum(DISTINCT x)`, `groupArray(DISTINCT x)`, `corrStableDistinct(DISTINCT x, y)` and so on.

## -OrDefault

Changes behavior of an aggregate function.

If an aggregate function does not have input values, with this combinator it returns the default value for its return data type. Applies to the aggregate functions that can take empty input data.

`-OrDefault` can be used with other combinators.

### Syntax

```
<aggFunction>OrDefault(x)
```

### Arguments

- `x` — Aggregate function parameters.

### Returned values

Returns the default value of an aggregate function's return type if there is nothing to aggregate.

Type depends on the aggregate function used.

### Example

Query:

```
SELECT avg(number), avgOrDefault(number) FROM numbers(0)
```

Result:

|             |   |                      |   |
|-------------|---|----------------------|---|
| avg(number) | — | avgOrDefault(number) | — |
| nan         |   | 0                    |   |
|             |   |                      |   |

Also `-OrDefault` can be used with another combinators. It is useful when the aggregate function does not accept the empty input.

Query:

```
SELECT avgOrDefaultIf(x, x > 10)
FROM
(
    SELECT toDecimal32(1.23, 2) AS x
)
```

Result:

```
avgOrDefaultIf(x, greater(x, 10))—
0.00 |
```

## -OrNull

Changes behavior of an aggregate function.

This combinator converts a result of an aggregate function to the **Nullable** data type. If the aggregate function does not have values to calculate it returns **NULL**.

-OrNull can be used with other combinators.

### Syntax

```
<aggFunction>OrNull(x)
```

### Arguments

- x — Aggregate function parameters.

### Returned values

- The result of the aggregate function, converted to the **Nullable** data type.
- NULL**, if there is nothing to aggregate.

Type: **Nullable(aggregate function return type)**.

### Example

Add `-orNull` to the end of aggregate function.

Query:

```
SELECT sumOrNull(number), toTypeName(sumOrNull(number)) FROM numbers(10) WHERE number > 10
```

Result:

```
sumOrNull(number)—toTypeName(sumOrNull(number))—
NULL | Nullable(UInt64) |
```

Also `-OrNull` can be used with another combinators. It is useful when the aggregate function does not accept the empty input.

Query:

```
SELECT avgOrNullIf(x, x > 10)
FROM
(
    SELECT toDecimal32(1.23, 2) AS x
)
```

Result:

```
avgOrNullIf(x, greater(x, 10))
```

NULL |

## -Resample

Lets you divide data into groups, and then separately aggregates the data in those groups. Groups are created by splitting the values from one column into intervals.

```
<aggFunction>Resample(start, end, step)(<aggFunction_params>, resampling_key)
```

### Arguments

- `start` — Starting value of the whole required interval for `resampling_key` values.
- `stop` — Ending value of the whole required interval for `resampling_key` values. The whole interval does not include the `stop` value [`start, stop`).
- `step` — Step for separating the whole interval into subintervals. The `aggFunction` is executed over each of those subintervals independently.
- `resampling_key` — Column whose values are used for separating data into intervals.
- `aggFunction_params` — `aggFunction` parameters.

### Returned values

- Array of `aggFunction` results for each subinterval.

### Example

Consider the `people` table with the following data:

| name   | age | wage |
|--------|-----|------|
| John   | 16  | 10   |
| Alice  | 30  | 15   |
| Mary   | 35  | 8    |
| Evelyn | 48  | 11.5 |
| David  | 62  | 9.9  |
| Brian  | 60  | 16   |

Let's get the names of the people whose age lies in the intervals of [30,60) and [60,75). Since we use integer representation for age, we get ages in the [30, 59] and [60,74] intervals.

To aggregate names in an array, we use the `groupArray` aggregate function. It takes one argument. In our case, it's the `name` column. The `groupArrayResample` function should use the `age` column to aggregate names by age. To define the required intervals, we pass the `30, 75, 30` arguments into the `groupArrayResample` function.

```
SELECT groupArrayResample(30, 75, 30)(name, age) FROM people
```

```
groupArrayResample(30, 75, 30)(name, age)———  
[['Alice','Mary','Evelyn'],['David','Brian']] |
```

Consider the results.

John is out of the sample because he's too young. Other people are distributed according to the specified age intervals.

Now let's count the total number of people and their average wage in the specified age intervals.

```
SELECT  
    countResample(30, 75, 30)(name, age) AS amount,  
    avgResample(30, 75, 30)(wage, age) AS avg_wage  
FROM people
```

```
amount——avg_wage———  
[3,2] | [11.5,12.949999809265137] |
```

## Parametric Aggregate Functions

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.

### histogram

Calculates an adaptive histogram. It does not guarantee precise results.

```
histogram(number_of_bins)(values)
```

The function uses [A Streaming Parallel Decision Tree Algorithm](#). The borders of histogram bins are adjusted as new data enters a function. In common case, the widths of bins are not equal.

#### Arguments

values — [Expression](#) resulting in input values.

#### Parameters

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.

#### Returned values

- **Array** of **Tuples** of the following format:

```
...
[ (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.
```

## Example

```
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)] |
```

You can visualize a histogram with the **bar** function, for example:

```
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
)
```



In this case, you should remember that you do not know the histogram bin borders.

## sequenceMatch(pattern)(timestamp, cond1, cond2, ...)

Checks whether the sequence contains an event chain that matches the pattern.

```
sequenceMatch(pattern)(timestamp, cond1, cond2, ...)
```

## Warning

Events that occur at the same second may lay in the sequence in an undefined order affecting the result.

## Arguments

- `timestamp` — Column considered to contain time data. Typical data types are `Date` and `DateTime`. You can also use any of the supported `UInt` data types.
- `cond1, cond2` — Conditions that describe the chain of events. Data type: `UInt8`. You can pass up to 32 condition arguments. The function takes only the events described in these conditions into account. If the sequence contains data that isn't described in a condition, the function skips them.

## Parameters

- `pattern` — Pattern string. See [Pattern syntax](#).

## Returned values

- 1, if the pattern is matched.
- 0, if the pattern isn't matched.

Type: `UInt8`.

## Pattern syntax

- `(?N)` — Matches the condition argument at position `N`. Conditions are numbered in the `[1, 32]` range. For example, `(?1)` matches the argument passed to the `cond1` parameter.
- `.*` — Matches any number of events. You do not 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)` matches events that occur more than 1800 seconds from each other. An arbitrary number of any events can lay between these events. You can use the `>=`, `>`, `<`, `<=`, `==` operators.

## Examples

Consider data in the `t` table:

| time | number |
|------|--------|
| 1    | 1      |
| 2    | 3      |
| 3    | 2      |

Perform the query:

```
SELECT sequenceMatch('(?1)(?2)')(time, number = 1, number = 2) FROM t
```

```
sequenceMatch('(?1)(?2)')(time, equals(number, 1), equals(number, 2))—
1 |
```

The function found the event chain where number 2 follows number 1. It skipped number 3 between them, because the number is not described as an event. If we want to take this number into account when searching for the event chain given in the example, we should make a condition for it.

```
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 |
```

In this case, the function couldn't find the event chain matching the pattern, because the event for number 3 occurred between 1 and 2. If in the same case we checked the condition for number 4, the sequence would match the pattern.

```
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 |
```

## See Also

- [sequenceCount](#)

## sequenceCount(pattern)(time, cond1, cond2, ...)

Counts the number of event chains that matched the pattern. The function searches event chains that do not overlap. It starts to search for the next chain after the current chain is matched.

## Warning

Events that occur at the same second may lay in the sequence in an undefined order affecting the result.

```
sequenceCount(pattern)(timestamp, cond1, cond2, ...)
```

## Arguments

- `timestamp` — Column considered to contain time data. Typical data types are `Date` and `DateTime`. You can also use any of the supported `UInt` data types.
- `cond1, cond2` — Conditions that describe the chain of events. Data type: `UInt8`. You can pass up to 32 condition arguments. The function takes only the events described in these conditions into account. If the sequence contains data that isn't described in a condition, the function skips them.

## Parameters

- `pattern` — Pattern string. See [Pattern syntax](#).

## Returned values

- Number of non-overlapping event chains that are matched.

Type: `UInt64`.

## Example

Consider data in the `t` table:

| time | number |
|------|--------|
| 1    | 1      |
| 2    | 3      |
| 3    | 2      |
| 4    | 1      |
| 5    | 3      |
| 6    | 2      |

Count how many times the number 2 occurs after the number 1 with any amount of other numbers between them:

```
SELECT sequenceCount('(?1).*(?2)')(time, number = 1, number = 2) FROM t
```

```
sequenceCount('(?1).*(?2)')(time, equals(number, 1), equals(number, 2))—  
2 |
```

## See Also

- [sequenceMatch](#)

## windowFunnel

Searches for event chains in a sliding time window and calculates the maximum number of events that occurred from the chain.

The function works according to the algorithm:

- The function searches for data that triggers the first condition in the chain and sets the event counter to 1. This is the moment when the sliding window starts.
- If events from the chain occur sequentially within the window, the counter is incremented. If the sequence of events is disrupted, the counter isn't incremented.
- If the data has multiple event chains at varying points of completion, the function will only output the size of the longest chain.

## Syntax

```
windowFunnel(window, [mode, [mode, ... ]])(timestamp, cond1, cond2, ..., condN)
```

## Arguments

- `timestamp` — Name of the column containing the timestamp. Data types supported: [Date](#), [DateTime](#) and other unsigned integer types (note that even though `timestamp` supports the `UInt64` type, its value can't exceed the `Int64` maximum, which is  $2^{63} - 1$ ).
- `cond` — Conditions or data describing the chain of events. [UInt8](#).

## Parameters

- `window` — Length of the sliding window, it is the time interval between the first and the last condition. The unit of `window` depends on the `timestamp` itself and varies. Determined using the expression `timestamp of cond1 <= timestamp of cond2 <= ... <= timestamp of condN <= timestamp of cond1 + window`.

- `mode` — It is an optional argument. One or more modes can be set.
  - `'strict_deduplication'` — If the same condition holds for the sequence of events, then such repeating event interrupts further processing.
  - `'strict_order'` — Don't allow interventions of other events. E.g. in the case of A->B->D->C, it stops finding A->B->C at the D and the max event level is 2.
  - `'strict_increase'` — Apply conditions only to events with strictly increasing timestamps.

## Returned value

The maximum number of consecutive triggered conditions from the chain within the sliding time window. All the chains in the selection are analyzed.

Type: Integer.

## Example

Determine if a set period of time is enough for the user to select a phone and purchase it twice in the online store.

Set the following chain of events:

1. The user logged in to their account on the store (`eventID = 1003`).
2. The user searches for a phone (`eventID = 1007, product = 'phone'`).
3. The user placed an order (`eventID = 1009`).
4. The user made the order again (`eventID = 1010`).

Input table:

| 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   |

Find out how far the user `user_id` could get through the chain in a period in January-February of 2019.

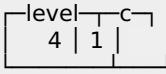
Query:

```

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;

```

Result:



## retention

The function takes as arguments a set of conditions from 1 to 32 arguments of type `UInt8` that indicate whether a certain condition was met for the event.

Any condition can be specified as an argument (as in **WHERE**).

The conditions, except the first, apply in pairs: the result of the second will be true if the first and second are true, of the third if the first and third are true, etc.

### Syntax

```
retention(cond1, cond2, ..., cond32);
```

### Arguments

- `cond` — An expression that returns a `UInt8` result (1 or 0).

### Returned value

The array of 1 or 0.

- 1 — Condition was met for the event.
- 0 — Condition wasn't met for the event.

Type: `UInt8`.

### Example

Let's consider an example of calculating the `retention` function to determine site traffic.

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);
```

Input table:

Query:

```
SELECT * FROM retention_test
```

Result:

| 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. Group users by unique ID uid using the retention function.

Query:

```
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
```

Result:

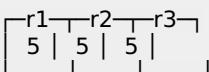
| uid | r       |
|-----|---------|
| 0   | [1,1,1] |
| 1   | [1,1,1] |
| 2   | [1,1,1] |
| 3   | [1,1,1] |
| 4   | [1,1,1] |
| 5   | [0,0,0] |
| 6   | [0,0,0] |
| 7   | [0,0,0] |
| 8   | [0,0,0] |
| 9   | [0,0,0] |
| 10  | [0,0,0] |
| 11  | [0,0,0] |
| 12  | [0,0,0] |
| 13  | [0,0,0] |
| 14  | [0,0,0] |

### 3. Calculate the total number of site visits per day.

Query:

```
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
)
```

Result:



Where:

- **r1**- the number of unique visitors who visited the site during 2020-01-01 (the `cond1` condition).
- **r2**- the number of unique visitors who visited the site during a specific time period between 2020-01-01 and 2020-01-02 (`cond1` and `cond2` conditions).
- **r3**- the number of unique visitors who visited the site during a specific time period between 2020-01-01 and 2020-01-03 (`cond1` and `cond3` conditions).

## 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.

Recommended for use with small Ns, up to 10. The maximum value of N is 100.

For the state of an aggregate function, it uses the amount of memory equal to  $1 + N * \text{the size of one value of bytes}$ .

For strings, it stores a non-cryptographic hash of 8 bytes. That is, the calculation is approximated for strings.

The function also works for several arguments.

It works as fast as possible, except for cases when a large N value is used and the number of unique values is slightly less than N.

Usage example:

```
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)(keys, values)

Same behavior as **sumMap** except that an array of keys is passed as a parameter. This can be especially useful when working with a high cardinality of keys.

# sequenceNextNode

Returns a value of the next event that matched an event chain.

*Experimental function, SET allow\_experimental\_funnel\_functions = 1 to enable it.*

## Syntax

```
sequenceNextNode(direction, base)(timestamp, event_column, base_condition, event1, event2, event3, ...)
```

## Parameters

- `direction` — Used to navigate to directions.
  - `forward` — Moving forward.
  - `backward` — Moving backward.
- `base` — Used to set the base point.
  - `head` — Set the base point to the first event.
  - `tail` — Set the base point to the last event.
  - `first_match` — Set the base point to the first matched `event1`.
  - `last_match` — Set the base point to the last matched `event1`.

## Arguments

- `timestamp` — Name of the column containing the timestamp. Data types supported: [Date](#), [DateTime](#) and other unsigned integer types.
- `event_column` — Name of the column containing the value of the next event to be returned. Data types supported: [String](#) and [Nullable\(String\)](#).
- `base_condition` — Condition that the base point must fulfill.
- `event1, event2, ...` — Conditions describing the chain of events. [UInt8](#).

## Returned values

- `event_column[next_index]` — If the pattern is matched and next value exists.
- `NULL` - If the pattern isn't matched or next value doesn't exist.

Type: [Nullable\(String\)](#).

## Example

It can be used when events are A->B->C->D->E and you want to know the event following B->C, which is D.

The query statement searching the event following A->B:

```

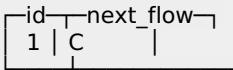
CREATE TABLE test_flow (
    dt DateTime,
    id int,
    page String)
ENGINE = MergeTree()
PARTITION BY toYYYYMMDD(dt)
ORDER BY id;

INSERT INTO test_flow VALUES (1, 1, 'A') (2, 1, 'B') (3, 1, 'C') (4, 1, 'D') (5, 1, 'E');

SELECT id, sequenceNextNode('forward', 'head')(dt, page, page = 'A', page = 'B') as next_flow FROM
test_flow GROUP BY id;

```

Result:



### Behavior for forward and head

```

ALTER TABLE test_flow DELETE WHERE 1 = 1 settings mutations_sync = 1;

INSERT INTO test_flow VALUES (1, 1, 'Home') (2, 1, 'Gift') (3, 1, 'Exit');
INSERT INTO test_flow VALUES (1, 2, 'Home') (2, 2, 'Home') (3, 2, 'Gift') (4, 2, 'Basket');
INSERT INTO test_flow VALUES (1, 3, 'Gift') (2, 3, 'Home') (3, 3, 'Gift') (4, 3, 'Basket');

```

```

SELECT id, sequenceNextNode('forward', 'head')(dt, page, page = 'Home', page = 'Home', page = 'Gift') FROM
test_flow GROUP BY id;

dt id page
1970-01-01 09:00:01 1 Home // Base point, Matched with Home
1970-01-01 09:00:02 1 Gift // Matched with Gift
1970-01-01 09:00:03 1 Exit // The result

1970-01-01 09:00:01 2 Home // Base point, Matched with Home
1970-01-01 09:00:02 2 Home // Unmatched with Gift
1970-01-01 09:00:03 2 Gift
1970-01-01 09:00:04 2 Basket

1970-01-01 09:00:01 3 Gift // Base point, Unmatched with Home
1970-01-01 09:00:02 3 Home
1970-01-01 09:00:03 3 Gift
1970-01-01 09:00:04 3 Basket

```

### Behavior for backward and tail

```

SELECT id, sequenceNextNode('backward', 'tail')(dt, page, page = 'Basket', page = 'Basket', page = 'Gift') FROM
test_flow GROUP BY id;

dt id page
1970-01-01 09:00:01 1 Home
1970-01-01 09:00:02 1 Gift
1970-01-01 09:00:03 1 Exit // Base point, Unmatched with Basket

1970-01-01 09:00:01 2 Home
1970-01-01 09:00:02 2 Home // The result
1970-01-01 09:00:03 2 Gift // Matched with Gift
1970-01-01 09:00:04 2 Basket // Base point, Matched with Basket

1970-01-01 09:00:01 3 Gift
1970-01-01 09:00:02 3 Home // The result
1970-01-01 09:00:03 3 Gift // Base point, Matched with Gift
1970-01-01 09:00:04 3 Basket // Base point, Matched with Basket

```

## Behavior for forward and first\_match

```
SELECT id, sequenceNextNode('forward', 'first_match')(dt, page, page = 'Gift', page = 'Gift') FROM test_flow GROUP BY id;

      dt  id  page
1970-01-01 09:00:01  1  Home
1970-01-01 09:00:02  1  Gift // Base point
1970-01-01 09:00:03  1  Exit // The result

1970-01-01 09:00:01  2  Home
1970-01-01 09:00:02  2  Home
1970-01-01 09:00:03  2  Gift // Base point
1970-01-01 09:00:04  2  Basket  The result

1970-01-01 09:00:01  3  Gift // Base point
1970-01-01 09:00:02  3  Home // The result
1970-01-01 09:00:03  3  Gift
1970-01-01 09:00:04  3  Basket
```

```
SELECT id, sequenceNextNode('forward', 'first_match')(dt, page, page = 'Gift', page = 'Gift', page = 'Home') FROM test_flow GROUP BY id;
```

```
      dt  id  page
1970-01-01 09:00:01  1  Home
1970-01-01 09:00:02  1  Gift // Base point
1970-01-01 09:00:03  1  Exit // Unmatched with Home

1970-01-01 09:00:01  2  Home
1970-01-01 09:00:02  2  Home
1970-01-01 09:00:03  2  Gift // Base point
1970-01-01 09:00:04  2  Basket // Unmatched with Home

1970-01-01 09:00:01  3  Gift // Base point
1970-01-01 09:00:02  3  Home // Matched with Home
1970-01-01 09:00:03  3  Gift // The result
1970-01-01 09:00:04  3  Basket
```

## Behavior for backward and last\_match

```
SELECT id, sequenceNextNode('backward', 'last_match')(dt, page, page = 'Gift', page = 'Gift') FROM test_flow GROUP BY id;

      dt  id  page
1970-01-01 09:00:01  1  Home // The result
1970-01-01 09:00:02  1  Gift // Base point
1970-01-01 09:00:03  1  Exit

1970-01-01 09:00:01  2  Home
1970-01-01 09:00:02  2  Home // The result
1970-01-01 09:00:03  2  Gift // Base point
1970-01-01 09:00:04  2  Basket

1970-01-01 09:00:01  3  Gift
1970-01-01 09:00:02  3  Home // The result
1970-01-01 09:00:03  3  Gift // Base point
1970-01-01 09:00:04  3  Basket
```

```

SELECT id, sequenceNextNode('backward', 'last_match')(dt, page, page = 'Gift', page = 'Gift', page = 'Home') FROM test_flow GROUP BY id;

    dt  id  page
1970-01-01 09:00:01  1  Home // Matched with Home, the result is null
1970-01-01 09:00:02  1  Gift // Base point
1970-01-01 09:00:03  1  Exit

1970-01-01 09:00:01  2  Home // The result
1970-01-01 09:00:02  2  Home // Matched with Home
1970-01-01 09:00:03  2  Gift // Base point
1970-01-01 09:00:04  2  Basket

1970-01-01 09:00:01  3  Gift // The result
1970-01-01 09:00:02  3  Home // Matched with Home
1970-01-01 09:00:03  3  Gift // Base point
1970-01-01 09:00:04  3  Basket

```

## Behavior for base\_condition

```

CREATE TABLE test_flow_basecond
(
    `dt` DateTime,
    `id` int,
    `page` String,
    `ref` String
)
ENGINE = MergeTree
PARTITION BY toYYYYMMDD(dt)
ORDER BY id;

INSERT INTO test_flow_basecond VALUES (1, 1, 'A', 'ref4') (2, 1, 'A', 'ref3') (3, 1, 'B', 'ref2') (4, 1, 'B', 'ref1');

```

```

SELECT id, sequenceNextNode('forward', 'head')(dt, page, ref = 'ref1', page = 'A') FROM test_flow_basecond GROUP BY id;

    dt  id  page  ref
1970-01-01 09:00:01  1  A    ref4 // The head can not be base point because the ref column of the head unmatched
                           with 'ref1'.
1970-01-01 09:00:02  1  A    ref3
1970-01-01 09:00:03  1  B    ref2
1970-01-01 09:00:04  1  B    ref1
```

```sql
SELECT id, sequenceNextNode('backward', 'tail')(dt, page, ref = 'ref4', page = 'B') FROM test_flow_basecond GROUP BY id;

    dt  id  page  ref
1970-01-01 09:00:01  1  A    ref4
1970-01-01 09:00:02  1  A    ref3
1970-01-01 09:00:03  1  B    ref2
1970-01-01 09:00:04  1  B    ref1 // The tail can not be base point because the ref column of the tail unmatched
                           with 'ref4'.

```

```

SELECT id, sequenceNextNode('forward', 'first_match')(dt, page, ref = 'ref3', page = 'A') FROM test_flow_basecond GROUP BY id;

```

```

    dt  id  page  ref
1970-01-01 09:00:01  1  A    ref4 // This row can not be base point because the ref column unmatched with 'ref3'.
1970-01-01 09:00:02  1  A    ref3 // Base point
1970-01-01 09:00:03  1  B    ref2 // The result
1970-01-01 09:00:04  1  B    ref1

```

```
SELECT id, sequenceNextNode('backward', 'last_match')(dt, page, ref = 'ref2', page = 'B') FROM test_flow_basesecond
GROUP BY id;
```

dt	id	page	ref
1970-01-01 09:00:01	1	A	ref4
1970-01-01 09:00:02	1	A	ref3 // The result
1970-01-01 09:00:03	1	B	ref2 // Base point
1970-01-01 09:00:04	1	B	ref1 // This row can not be base point because the ref column unmatched with 'ref2'.

## Table Functions

Table functions are methods for constructing tables.

You can use table functions in:

- **FROM** clause of the **SELECT** query.

The method for creating a temporary table that is available only in the current query. The table is deleted when the query finishes.

- **CREATE TABLE AS table\_function()** query.

It's one of the methods of creating a table.

### Warning

You can't use table functions if the **allow\_ddl** setting is disabled.

Function	Description
file	Creates a <b>File</b> -engine table.
merge	Creates a <b>Merge</b> -engine table.
numbers	Creates a table with a single column filled with integer numbers.
remote	Allows you to access remote servers without creating a <b>Distributed</b> -engine table.
url	Creates a <b>Url</b> -engine table.
mysql	Creates a <b>MySQL</b> -engine table.
postgresql	Creates a <b>PostgreSQL</b> -engine table.
jdbc	Creates a <b>JDBC</b> -engine table.
odbc	Creates a <b>ODBC</b> -engine table.
hdfs	Creates a <b>HDFS</b> -engine table.
s3	Creates a <b>S3</b> -engine table.

Function	Description
sqlite	Creates a <code>sqlite</code> -engine table.

## file

Creates a table from a file. This table function is similar to `url` and `hdfs` ones.

`file` function can be used in `SELECT` and `INSERT` queries on data in `File` tables.

### Syntax

```
file(path, format, structure)
```

### Parameters

- `path` — The relative path to the file from `user_files_path`. Path to file support following globs in read-only mode: `*`, `?`, `{abc,def}` and `{N..M}` where `N, M` — numbers, `'abc', 'def'` — strings.
- `format` — The `format` of the file.
- `structure` — Structure of the table. Format: `'column1_name column1_type, column2_name column2_type, ...'`.

### Returned value

A table with the specified structure for reading or writing data in the specified file.

### Examples

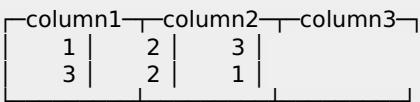
Setting `user_files_path` and the contents of the file `test.csv`:

```
$ 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
```

Getting data from a table in `test.csv` and selecting the first two rows from it:

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



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;
```

Inserting data from a file into a table:

```
INSERT INTO FUNCTION file('test.csv', 'CSV', 'column1 UInt32, column2 UInt32, column3 UInt32') VALUES (1, 2, 3), (3, 2, 1);
SELECT * FROM file('test.csv', 'CSV', 'column1 UInt32, column2 UInt32, column3 UInt32');
```

column1	column2	column3
1	2	3
3	2	1

## Globs in Path

Multiple path components can have globs. For being processed file must exist and match to the whole path pattern (not only suffix or prefix).

- `*` — Substitutes any number of any characters except/ including empty string.
- `?` — 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.

Constructions with `{}` are similar to the `remote` table function.

### Example

Suppose we have several files with the following relative paths:

- `'some_dir/some_file_1'`
- `'some_dir/some_file_2'`
- `'some_dir/some_file_3'`
- `'another_dir/some_file_1'`
- `'another_dir/some_file_2'`
- `'another_dir/some_file_3'`

Query the number of rows in these files:

```
SELECT count(*) FROM file('{some,another}_dir/some_file_{1..3}', 'TSV', 'name String, value UInt32');
```

Query the number of rows in all files of these two directories:

```
SELECT count(*) FROM file('{some,another}_dir/*', 'TSV', 'name String, value UInt32');
```

## Warning

If your listing of files contains number ranges with leading zeros, use the construction with braces for each digit separately or use `?`.

### Example

Query the data from files named `file000`, `file001`, ... , `file999`:

```
SELECT count(*) FROM file('big_dir/file{0..9}{0..9}{0..9}', 'CSV', 'name String, value UInt32');
```

## Virtual Columns

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

### See Also

- [Virtual columns](#)

## merge

`merge(db_name, 'tables_regexp')` – Creates a temporary Merge table. For more information, see the section “Table engines, Merge”.

The table structure is taken from the first table encountered that matches the regular expression.

## numbers

`numbers(N)` – Returns a table with the single ‘number’ column (UInt64) that contains integers from 0 to N-1.  
`numbers(N, M)` - Returns a table with the single ‘number’ column (UInt64) that contains integers from N to (N + M - 1).

Similar to the `system.numbers` table, it can be used for testing and generating successive values, `numbers(N, M)` more efficient than `system.numbers`.

The following queries are equivalent:

```
SELECT * FROM numbers(10);
SELECT * FROM numbers(0, 10);
SELECT * FROM system.numbers LIMIT 10;
```

Examples:

```
-- Generate a sequence of dates from 2010-01-01 to 2010-12-31
select toDate('2010-01-01') + number as d FROM numbers(365);
```

## remote, remoteSecure

Allows to access remote servers without creating a [Distributed](#) table. `remoteSecure` - same as `remote` but with a secured connection.

Both functions can be used in `SELECT` and `INSERT` queries.

### Syntax

```
remote('addresses_expr', db, table[, 'user'][, 'password'], sharding_key[])
remote('addresses_expr', db.table[, 'user'][, 'password'], sharding_key[])
remoteSecure('addresses_expr', db, table[, 'user'][, 'password'], sharding_key[])
remoteSecure('addresses_expr', db.table[, 'user'][, 'password'], sharding_key[])
```

## Parameters

- `addresses_expr` — An expression that generates addresses of remote servers. This may be just one server address. The server address is `host:port`, or just `host`.

The host can be specified as the server name, or as the IPv4 or IPv6 address. An IPv6 address is specified in square brackets.

The port is the TCP port on the remote server. If the port is omitted, it uses `tcp_port` from the server's config file in `remote` (by default, 9000) and `tcp_port_secure` in `remoteSecure` (by default, 9440).

The port is required for an IPv6 address.

Type: `String`.

- `db` — Database name. Type: `String`.
- `table` — Table name. Type: `String`.
- `user` — User name. If the user is not specified, `default` is used. Type: `String`.
- `password` — User password. If the password is not specified, an empty password is used. Type: `String`.
- `sharding_key` — Sharding key to support distributing data across nodes. For example: `insert into remote('127.0.0.1:9000,127.0.0.2', db, table, 'default', rand())`. Type: `UInt32`.

## Returned value

The dataset from remote servers.

## Usage

Using the `remote` table function is less optimal than creating a `Distributed` table because in this case the server connection is re-established for every request. Also, if hostnames are set, the names are resolved, and errors are not counted when working with various replicas. When processing a large number of queries, always create the `Distributed` table ahead of time, and do not use the `remote` table function.

The `remote` table function can be useful in the following cases:

- Accessing a specific server for data comparison, debugging, and testing.
- Queries between various ClickHouse clusters for research purposes.
- Infrequent distributed requests that are made manually.
- Distributed requests where the set of servers is re-defined each time.

## Addresses

```
example01-01-1
example01-01-1:9000
localhost
127.0.0.1
[::]:9000
[2a02:6b8:0:1111::11]:9000
```

Multiple addresses can be comma-separated. In this case, ClickHouse will use distributed processing, so it will send the query to all specified addresses (like shards with different data). Example:

```
example01-01-1,example01-02-1
```

## Examples

Selecting data from a remote server:

```
SELECT * FROM remote('127.0.0.1', db.remote_engine_table) LIMIT 3;
```

Inserting data from a remote server into a table:

```
CREATE TABLE remote_table (name String, value UInt32) ENGINE=Memory;
INSERT INTO FUNCTION remote('127.0.0.1', currentDatabase(), 'remote_table') VALUES ('test', 42);
SELECT * FROM remote_table;
```

## Globs in Addresses

Patterns in curly brackets { } are used to generate a set of shards and to specify replicas. If there are multiple pairs of curly brackets, then the direct product of the corresponding sets is generated. The following pattern types are supported.

- {*a,b*} - Any number of variants separated by a comma. The pattern is replaced with *a* in the first shard address and it is replaced with *b* in the second shard address and so on. For instance, `example0{1,2}-1` generates addresses `example01-1` and `example02-1`.
- {*n..m*} - A range of numbers. This pattern generates shard addresses with incrementing indices from *n* to *m*. `example0{1..2}-1` generates `example01-1` and `example02-1`.
- {*0n..0m*} - A range of numbers with leading zeroes. This modification preserves leading zeroes in indices. The pattern `example{01..03}-1` generates `example01-1`, `example02-1` and `example03-1`.
- {*a|b*} - Any number of variants separated by a |. The pattern specifies replicas. For instance, `example01-{1|2}` generates replicas `example01-1` and `example01-2`.

The query will be sent to the first healthy replica. However, for `remote` the replicas are iterated in the order currently set in the [load\\_balancing](#) setting.

The number of generated addresses is limited by [table\\_function\\_remote\\_max\\_addresses](#) setting.

## url

`url` function creates a table from the URL with given `format` and `structure`.

`url` function may be used in `SELECT` and `INSERT` queries on data in [URL](#) tables.

### Syntax

```
url(URL, format, structure)
```

### Parameters

- `URL` — HTTP or HTTPS server address, which can accept `GET` or `POST` requests (for `SELECT` or `INSERT` queries correspondingly). Type: [String](#).
- `format` — [Format](#) of the data. Type: [String](#).
- `structure` — Table structure in '`UserID UInt64, Name String`' format. Determines column names and types. Type: [String](#).

### Returned value

A table with the specified format and structure and with data from the defined URL.

## Examples

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;
```

Inserting data from a URL into a table:

```
CREATE TABLE test_table (column1 String, column2 UInt32) ENGINE=Memory;
INSERT INTO FUNCTION url('http://127.0.0.1:8123/?query=INSERT+INTO+test_table+FORMAT+CSV', 'CSV', 'column1 String, column2 UInt32') VALUES ('http interface', 42);
SELECT * FROM test_table;
```

## Globs in URL

Patterns in curly brackets {} are used to generate a set of shards or to specify failover addresses.

Supported pattern types and examples see in the description of the [remote](#) function.

Character | inside patterns is used to specify failover addresses. They are iterated in the same order as listed in the pattern. The number of generated addresses is limited by [glob\\_expansion\\_max\\_elements](#) setting.

## mysql

Allows SELECT and INSERT queries to be performed on data that is stored on a remote MySQL server.

### Syntax

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

### Arguments

- `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` queries to `REPLACE INTO`. Possible values:
  - `0` - The query is executed as `INSERT INTO`.
  - `1` - The query is executed as `REPLACE INTO`.
- `on_duplicate_clause` — The `ON DUPLICATE KEY on_duplicate_clause` expression that is added to the `INSERT` query. Can be specified only with `replace_query = 0` (if you simultaneously pass `replace_query = 1` and `on_duplicate_clause`, ClickHouse generates an exception).

Example: `INSERT INTO t (c1,c2) VALUES ('a', 2) ON DUPLICATE KEY UPDATE c2 = c2 + 1;`

`on_duplicate_clause` here is `UPDATE c2 = c2 + 1`. See the MySQL documentation to find which `on_duplicate_clause` you can use with the `ON DUPLICATE KEY` clause.

Simple WHERE clauses such as `=`, `!=`, `>`, `>=`, `<`, `<=` are currently executed on the MySQL server.

The rest of the conditions and the LIMIT sampling constraint are executed in ClickHouse only after the query to MySQL finishes.

Supports multiple replicas that must be listed by |. For example:

```
SELECT name FROM mysql(`mysql{1|2|3}:3306`, 'mysql_database', 'mysql_table', 'user', 'password');
```

or

```
SELECT name FROM mysql(`mysql1:3306|mysql2:3306|mysql3:3306`, 'mysql_database', 'mysql_table', 'user', 'password');
```

## Returned Value

A table object with the same columns as the original MySQL table.

## Note

In the INSERT query to distinguish table function `mysql(...)` from table name with column names list, you must use keywords FUNCTION or TABLE FUNCTION. See examples below.

## Examples

Table in MySQL:

```
mysql> CREATE TABLE `test`.`test` (
    -> `int_id` INT NOT NULL AUTO_INCREMENT,
    -> `float` FLOAT NOT NULL,
    -> PRIMARY KEY (`int_id`));

mysql> INSERT INTO test (`int_id`, `float`) VALUES (1,2);

mysql> SELECT * FROM test;
+-----+-----+
| int_id | float |
+-----+-----+
|     1 |     2 |
+-----+-----+
```

Selecting data from ClickHouse:

```
SELECT * FROM mysql('localhost:3306', 'test', 'test', 'bayonet', '123');
```

int_id	float
1	2

Replacing and inserting:

```
INSERT INTO FUNCTION mysql('localhost:3306', 'test', 'test', 'bayonet', '123', 1) (int_id, float) VALUES (1, 3);
INSERT INTO TABLE FUNCTION mysql('localhost:3306', 'test', 'test', 'bayonet', '123', 0, 'UPDATE int_id = int_id + 1')
(int_id, float) VALUES (1, 4);
SELECT * FROM mysql('localhost:3306', 'test', 'test', 'bayonet', '123');
```

int_id	float
1	3
2	4

## See Also

- [The 'MySQL' table engine](#)
- [Using MySQL as a source of external dictionary](#)

# postgresql

Allows `SELECT` and `INSERT` queries to be performed on data that is stored on a remote PostgreSQL server.

## Syntax

```
postgresql('host:port', 'database', 'table', 'user', 'password'[, `schema`])
```

## Arguments

- `host:port` — PostgreSQL server address.
- `database` — Remote database name.
- `table` — Remote table name.
- `user` — PostgreSQL user.
- `password` — User password.
- `schema` — Non-default table schema. Optional.

## Returned Value

A table object with the same columns as the original PostgreSQL table.

## Note

In the `INSERT` query to distinguish table function `postgresql(...)` from table name with column names list you must use keywords `FUNCTION` or `TABLE FUNCTION`. See examples below.

## Implementation Details

`SELECT` queries on PostgreSQL side run as `COPY (SELECT ...)` TO `STDOUT` inside read-only PostgreSQL transaction with commit after each `SELECT` query.

Simple `WHERE` clauses such as `=`, `!=`, `>`, `>=`, `<`, `<=`, and `IN` are executed on the PostgreSQL server.

All joins, aggregations, sorting, `IN [ array ]` conditions and the `LIMIT` sampling constraint are executed in ClickHouse only after the query to PostgreSQL finishes.

`INSERT` queries on PostgreSQL side run as `COPY "table_name" (field1, field2, ... fieldN) FROM STDIN` inside PostgreSQL transaction with auto-commit after each `INSERT` statement.

PostgreSQL Array types converts into ClickHouse arrays.

## Note

Be careful, in PostgreSQL an array data type column like Integer[] may contain arrays of different dimensions in different rows, but in ClickHouse it is only allowed to have multidimensional arrays of the same dimension in all rows.

Supports multiple replicas that must be listed by |. For example:

```
SELECT name FROM postgres(`postgres{1|2|3}:5432`, 'postgres_database', 'postgres_table', 'user', 'password');
```

or

```
SELECT name FROM postgres(`postgres1:5431|postgres2:5432`, 'postgres_database', 'postgres_table', 'user', 'password');
```

Supports replicas priority for PostgreSQL dictionary source. The bigger the number in map, the less the priority. The highest priority is 0.

## Examples

Table in PostgreSQL:

```
postgres=# CREATE TABLE "public"."test" (
  "int_id" SERIAL,
  "int_nullable" INT NULL DEFAULT NULL,
  "float" FLOAT NOT NULL,
  "str" VARCHAR(100) NOT NULL DEFAULT '',
  "float_nullable" FLOAT NULL DEFAULT NULL,
  PRIMARY KEY (int_id));

CREATE TABLE

postgres=# INSERT INTO test (int_id, str, "float") VALUES (1,'test',2);
INSERT 0 1

postgres> SELECT * FROM test;
 int_id | int_nullable | float | str  | float_nullable
-----+-----+-----+-----+
   1    |      NULL    |     2 | test |      NULL
(1 row)
```

Selecting data from ClickHouse:

```
SELECT * FROM postgres('localhost:5432', 'test', 'test', 'postgresql_user', 'password') WHERE str IN ('test');
```

int_id	int_nullable	float	str	float_nullable
1	NULL	2	test	NULL

Inserting:

```
INSERT INTO TABLE FUNCTION postgres('localhost:5432', 'test', 'test', 'postgresql_user', 'password') (int_id, float)
VALUES (2, 3);
SELECT * FROM postgres('localhost:5432', 'test', 'test', 'postgresql_user', 'password');
```

int_id	int_nullable	float	str	float_nullable
1	NULL	2	test	NULL
2	NULL	3		NULL

Using Non-default Schema:

```
postgres=# CREATE SCHEMA "nice.schema";
postgres=# CREATE TABLE "nice.schema"."nice.table" (a integer);
postgres=# INSERT INTO "nice.schema"."nice.table" SELECT i FROM generate_series(0, 99) as t(i)
```

```
CREATE TABLE pg_table_schema_with_dots (a UInt32)
ENGINE PostgreSQL('localhost:5432', 'clickhouse', 'nice.table', 'postgrsql_user', 'password', 'nice.schema');
```

## See Also

- [The PostgreSQL table engine](#)
- [Using PostgreSQL as a source of external dictionary](#)

## jdbc

`jdbc(datasource, schema, table)` - returns table that is connected via JDBC driver.

This table function requires separate [clickhouse-jdbc-bridge](#) program to be running.  
It supports Nullable types (based on DDL of remote table that is queried).

## Examples

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

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

```
SELECT * FROM jdbc('mysql-dev?p1=233', 'num Int32', 'select toInt32OrZero("") as num')
```

```
SELECT *
FROM jdbc('mysql-dev?p1=233', 'num Int32', 'select toInt32OrZero("") as num')
```

```
SELECT a.datasource AS server1, b.datasource AS server2, b.name AS db
FROM jdbc('mysql-dev?datasource_column', 'show databases') a
INNER JOIN jdbc('self?datasource_column', 'show databases') b ON a.Database = b.name
```

## odbc

Returns table that is connected via [ODBC](#).

```
odbc(connection_settings, external_database, external_table)
```

Parameters:

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

To safely implement ODBC connections, ClickHouse uses a separate program `clickhouse-odbc-bridge`. If the ODBC driver is loaded directly from `clickhouse-server`, driver problems can crash the ClickHouse server. ClickHouse automatically starts `clickhouse-odbc-bridge` when it is required. The ODBC bridge program is installed from the same package as the `clickhouse-server`.

The fields with the `NULL` values from the external table are converted into the default values for the base data type. For example, if a remote MySQL table field has the `INT NULL` type it is converted to 0 (the default value for ClickHouse `Int32` data type).

## Usage Example

### Getting data from the local MySQL installation via ODBC

This example is checked for Ubuntu Linux 18.04 and MySQL server 5.7.

Ensure that unixODBC and MySQL Connector are installed.

By default (if installed from packages), ClickHouse starts as user `clickhouse`. Thus you need to create and configure this user in the MySQL server.

```
$ sudo mysql
```

```
mysql> CREATE USER 'clickhouse'@'localhost' IDENTIFIED BY 'clickhouse';
mysql> GRANT ALL PRIVILEGES ON *.* TO 'clickhouse'@'clickhouse' WITH GRANT OPTION;
```

Then configure the connection in `/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
```

You can check the connection using the `isql` utility from the unixODBC installation.

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

Table in MySQL:

```
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)
```

Retrieving data from the MySQL table in ClickHouse:

```
SELECT * FROM odbc('DSN=mysqlconn', 'test', 'test')
```

int_id	int_nullable	float	float_nullable
1	0	2	0

## See Also

- [ODBC external dictionaries](#)
- [ODBC table engine](#).

## hdfs

Creates a table from files in HDFS. This table function is similar to [url](#) and [file](#) ones.

```
hdfs(URI, format, structure)
```

### Input parameters

- **URI** — The relative URI to the file in HDFS. Path to file support following globs in readonly mode: `*`, `?`, `{abc,def}` and `{N..M}` where `N, M` — numbers, `'abc', 'def'` — strings.
- **format** — The [format](#) of the file.
- **structure** — Structure of the table. Format `'column1_name column1_type, column2_name column2_type, ...'`.

### Returned value

A table with the specified structure for reading or writing data in the specified file.

### Example

Table from `hdfs://hdfs1:9000/test` and selection of the first two rows from it:

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

column1	column2	column3
1	2	3
3	2	1

## Globs in path

Multiple path components can have globs. For being processed file should exists and matches to the whole path pattern (not only suffix or prefix).

- \* — Substitutes any number of any characters except/ including empty string.
- ? — 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.

Constructions with {} are similar to the [remote table function](#)).

## Example

1. Suppose that we have several files with following URIs on HDFS:

- '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'

2. Query the amount of rows in these files:

```
SELECT count(*)
FROM hdfs('hdfs://hdfs1:9000/{some,another}_dir/some_file_{1..3}', 'TSV', 'name String, value UInt32')
```

3. Query the amount of rows in all files of these two directories:

```
SELECT count(*)
FROM hdfs('hdfs://hdfs1:9000/{some,another}_dir/*', 'TSV', 'name String, value UInt32')
```

## Warning

If your listing of files contains number ranges with leading zeros, use the construction with braces for each digit separately or use ?.

## Example

Query the data from files named file000, file001, ... , file999:

```
SELECT count(*)
FROM hdfs('hdfs://hdfs1:9000/big_dir/file{0..9}{0..9}{0..9}', 'CSV', 'name String, value UInt32')
```

## Virtual Columns

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

### See Also

- [Virtual columns](#)

## s3 Table Function

Provides table-like interface to select/insert files in [Amazon S3](#). This table function is similar to [hdfs](#), but provides S3-specific features.

### Syntax

```
s3(path, [aws_access_key_id, aws_secret_access_key,] format, structure, [compression])
```

### Arguments

- `path` — Bucket url with path to file. Supports following wildcards in readonly mode: `*`, `?`, `{abc,def}` and `{N..M}` where `N, M` — numbers, `'abc'`, `'def'` — strings. For more information see [here](#).
- `format` — The [format](#) of the file.
- `structure` — Structure of the table. Format `'column1_name column1_type, column2_name column2_type, ...'`.
- `compression` — Parameter is optional. Supported values: `none`, `gzip/gz`, `brotli/br`, `xz/LZMA`, `zstd/zst`. By default, it will autodetect compression by file extension.

### Returned value

A table with the specified structure for reading or writing data in the specified file.

### Examples

Selecting the first two rows from the table from S3 file <https://storage.yandexcloud.net/my-test-bucket-768/data.csv>:

```
SELECT *
FROM s3('https://storage.yandexcloud.net/my-test-bucket-768/data.csv', 'CSV', 'column1 UInt32, column2 UInt32,
column3 UInt32')
LIMIT 2;
```

column1	column2	column3
1	2	3
3	2	1

The similar but from file with `gzip` compression:

```
SELECT *
FROM s3('https://storage.yandexcloud.net/my-test-bucket-768/data.csv.gz', 'CSV', 'column1 UInt32, column2 UInt32,
column3 UInt32', 'gzip')
LIMIT 2;
```

column1	column2	column3
1	2	3
3	2	1

## Usage

Suppose that we have several files with following URIs on S3:

- '[https://storage.yandexcloud.net/my-test-bucket-768/some\\_prefix/some\\_file\\_1.csv](https://storage.yandexcloud.net/my-test-bucket-768/some_prefix/some_file_1.csv)'
- '[https://storage.yandexcloud.net/my-test-bucket-768/some\\_prefix/some\\_file\\_2.csv](https://storage.yandexcloud.net/my-test-bucket-768/some_prefix/some_file_2.csv)'
- '[https://storage.yandexcloud.net/my-test-bucket-768/some\\_prefix/some\\_file\\_3.csv](https://storage.yandexcloud.net/my-test-bucket-768/some_prefix/some_file_3.csv)'
- '[https://storage.yandexcloud.net/my-test-bucket-768/some\\_prefix/some\\_file\\_4.csv](https://storage.yandexcloud.net/my-test-bucket-768/some_prefix/some_file_4.csv)'
- '[https://storage.yandexcloud.net/my-test-bucket-768/another\\_prefix/some\\_file\\_1.csv](https://storage.yandexcloud.net/my-test-bucket-768/another_prefix/some_file_1.csv)'
- '[https://storage.yandexcloud.net/my-test-bucket-768/another\\_prefix/some\\_file\\_2.csv](https://storage.yandexcloud.net/my-test-bucket-768/another_prefix/some_file_2.csv)'
- '[https://storage.yandexcloud.net/my-test-bucket-768/another\\_prefix/some\\_file\\_3.csv](https://storage.yandexcloud.net/my-test-bucket-768/another_prefix/some_file_3.csv)'
- '[https://storage.yandexcloud.net/my-test-bucket-768/another\\_prefix/some\\_file\\_4.csv](https://storage.yandexcloud.net/my-test-bucket-768/another_prefix/some_file_4.csv)'

Count the amount of rows in files ending with numbers from 1 to 3:

```
SELECT count(*)
FROM s3('https://storage.yandexcloud.net/my-test-bucket-768/{some,another}_prefix/some_file_{1..3}.csv', 'CSV',
'name String, value UInt32')
```

count()
18

Count the total amount of rows in all files in these two directories:

```
SELECT count(*)
FROM s3('https://storage.yandexcloud.net/my-test-bucket-768/{some,another}_prefix/*', 'CSV', 'name String, value
UInt32')
```

count()
24

## Warning

If your listing of files contains number ranges with leading zeros, use the construction with braces for each digit separately or use ?.

Count the total amount of rows in files named file-000.csv, file-001.csv, ... , file-999.csv:

```
SELECT count(*)  
FROM s3('https://storage.yandexcloud.net/my-test-bucket-768/big_prefix/file-{000..999}.csv', 'CSV', 'name String,  
value UInt32');
```

```
count()  
12 |
```

Insert data into file test-data.csv.gz:

```
INSERT INTO FUNCTION s3('https://storage.yandexcloud.net/my-test-bucket-768/test-data.csv.gz', 'CSV', 'name String,  
value UInt32', 'gzip')  
VALUES ('test-data', 1), ('test-data-2', 2);
```

Insert data into file test-data.csv.gz from existing table:

```
INSERT INTO FUNCTION s3('https://storage.yandexcloud.net/my-test-bucket-768/test-data.csv.gz', 'CSV', 'name String,  
value UInt32', 'gzip')  
SELECT name, value FROM existing_table;
```

## Partitioned Write

If you specify `PARTITION BY` expression when inserting data into `S3` table, a separate file is created for each partition value. Splitting the data into separate files helps to improve reading operations efficiency.

### Examples

1. Using partition ID in a key creates separate files:

```
INSERT INTO TABLE FUNCTION  
s3('http://bucket.amazonaws.com/my_bucket/file_{_partition_id}.csv', 'CSV', 'a String, b UInt32, c UInt32')  
PARTITION BY a VALUES ('x', 2, 3), ('x', 4, 5), ('y', 11, 12), ('y', 13, 14), ('z', 21, 22), ('z', 23, 24);
```

As a result, the data is written into three files: `file_x.csv`, `file_y.csv`, and `file_z.csv`.

2. Using partition ID in a bucket name creates files in different buckets:

```
INSERT INTO TABLE FUNCTION  
s3('http://bucket.amazonaws.com/my_bucket_{_partition_id}/file.csv', 'CSV', 'a UInt32, b UInt32, c UInt32')  
PARTITION BY a VALUES (1, 2, 3), (1, 4, 5), (10, 11, 12), (10, 13, 14), (20, 21, 22), (20, 23, 24);
```

As a result, the data is written into three files in different buckets: `my_bucket_1/file.csv`, `my_bucket_10/file.csv`, and `my_bucket_20/file.csv`.

### See Also

- [S3 engine](#)

## input

`input(structure)` - table function that allows effectively convert and insert data sent to the server with given structure to the table with another structure.

`structure` - structure of data sent to the server in following format '`column1_name column1_type, column2_name column2_type, ...`'.

For example, '`id UInt32, name String`'.

This function can be used only in `INSERT SELECT` query and only once but otherwise behaves like ordinary table function

(for example, it can be used in subquery, etc.).

Data can be sent in any way like for ordinary `INSERT` query and passed in any available **format** that must be specified in the end of query (unlike ordinary `INSERT SELECT`).

The main feature of this function is that when server receives data from client it simultaneously converts it according to the list of expressions in the `SELECT` clause and inserts into the target table. Temporary table with all transferred data is not created.

## Examples

- Let the `test` table has the following structure (`a String, b String`) and data in `data.csv` has a different structure (`col1 String, col2 Date, col3 Int32`). Query for insert data from the `data.csv` into the `test` table with simultaneous conversion looks like this:

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

- If `data.csv` contains data of the same structure `test_structure` as the table `test` then these two queries are equal:

```
$ 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

Generates random data with given schema.

Allows to populate test tables with data.

Supports all data types that can be stored in table except `LowCardinality` and `AggregateFunction`.

```
generateRandom('name TypeName[, name TypeName]...', [, 'random_seed'][, 'max_string_length'][, 'max_array_length'])
```

## Arguments

- `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 `NULL` — seed is randomly generated.

## Returned Value

A table object with requested schema.

# Usage Example

```
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')

## cluster, clusterAllReplicas

Allows to access all shards in an existing cluster which configured in `remote_servers` section without creating a **Distributed** table. One replica of each shard is queried.

`clusterAllReplicas` function — same as `cluster`, but all replicas are queried. Each replica in a cluster is used as a separate shard/connection.

### Note

All available clusters are listed in the **system.clusters** table.

### Syntax

```
cluster('cluster_name', db.table[, sharding_key])
cluster('cluster_name', db, table[, sharding_key])
clusterAllReplicas('cluster_name', db.table[, sharding_key])
clusterAllReplicas('cluster_name', db, table[, sharding_key])
```

### Arguments

- `cluster_name` – Name of a cluster that is used to build a set of addresses and connection parameters to remote and local servers.
- `db.table` or `db, table` - Name of a database and a table.
- `sharding_key` - A sharding key. Optional. Needs to be specified if the cluster has more than one shard.

### Returned value

The dataset from clusters.

### Using Macros

`cluster_name` can contain macros — substitution in curly brackets. The substituted value is taken from the **macros** section of the server configuration file.

Example:

```
SELECT * FROM cluster('{cluster}', default.example_table);
```

### Usage and Recommendations

Using the `cluster` and `clusterAllReplicas` table functions are less efficient than creating a **Distributed** table because in this case, the server connection is re-established for every request. When processing a large number of queries, please always create the **Distributed** table ahead of time, and do not use the `cluster` and `clusterAllReplicas` table functions.

The `cluster` and `clusterAllReplicas` table functions can be useful in the following cases:

- Accessing a specific cluster for data comparison, debugging, and testing.
- Queries to various ClickHouse clusters and replicas for research purposes.
- Infrequent distributed requests that are made manually.

Connection settings like host, port, user, password, compression, secure are taken from `<remote_servers>` config section. See details in [Distributed engine](#).

## See Also

- [skip\\_unavailable\\_shards](#)
- [load\\_balancing](#)

# view

Turns a subquery into a table. The function implements views (see [CREATE VIEW](#)). The resulting table does not store data, but only stores the specified `SELECT` query. When reading from the table, ClickHouse executes the query and deletes all unnecessary columns from the result.

## Syntax

```
view(subquery)
```

## Arguments

- `subquery` — `SELECT` query.

## Returned value

- A table.

## Example

Input table:

id	name	days
1	January	31
2	February	29
3	March	31
4	April	30

Query:

```
SELECT * FROM view(SELECT name FROM months);
```

Result:

name
January
February
March
April

You can use the `view` function as a parameter of the `remote` and `cluster` table functions:

```
SELECT * FROM remote(`127.0.0.1`, view(SELECT a, b, c FROM table_name));
```

```
SELECT * FROM cluster(`cluster_name`, view(SELECT a, b, c FROM table_name));
```

## See Also

- [View Table Engine](#)

# null

Creates a temporary table of the specified structure with the `Null` table engine. According to the Null-engine properties, the table data is ignored and the table itself is immediately dropped right after the query execution. The function is used for the convenience of test writing and demonstrations.

## Syntax

```
null('structure')
```

## Parameter

- `structure` — A list of columns and column types. [String](#).

## Returned value

A temporary Null-engine table with the specified structure.

## Example

Query with the `null` function:

```
INSERT INTO function null('x UInt64') SELECT * FROM numbers_mt(1000000000);
```

can replace three queries:

```
CREATE TABLE t (x UInt64) ENGINE = Null;
INSERT INTO t SELECT * FROM numbers_mt(1000000000);
DROP TABLE IF EXISTS t;
```

See also:

- [Null table engine](#)

# dictionary

Displays the `dictionary` data as a ClickHouse table. Works the same way as `Dictionary` engine.

## Syntax

```
dictionary('dict')
```

## Arguments

- `dict` — A dictionary name. [String](#).

## Returned value

A ClickHouse table.

## Example

Input table `dictionary_source_table`:

id	value
0	0
1	1

Create a dictionary:

```
CREATE DICTIONARY new_dictionary(id UInt64, value UInt64 DEFAULT 0) PRIMARY KEY id
SOURCE(CLICKHOUSE(HOST 'localhost' PORT tcpPort() USER 'default' TABLE 'dictionary_source_table'))
LAYOUT(DIRECT());
```

Query:

```
SELECT * FROM dictionary('new_dictionary');
```

Result:

id	value
0	0
1	1

## See Also

- [Dictionary engine](#)

# s3Cluster Table Function

Allows processing files from [Amazon S3](#) in parallel from many nodes in a specified cluster. On initiator it creates a connection to all nodes in the cluster, discloses asterics in S3 file path, and dispatches each file dynamically. On the worker node it asks the initiator about the next task to process and processes it. This is repeated until all tasks are finished.

## Syntax

```
s3Cluster(cluster_name, source, [access_key_id, secret_access_key,] format, structure)
```

## Arguments

- `cluster_name` — Name of a cluster that is used to build a set of addresses and connection parameters to remote and local servers.
- `source` — URL to a file or a bunch of files. Supports following wildcards in readonly mode: `*`, `?`, `{'abc','def'}` and `{N..M}` where `N, M` — numbers, `abc, def` — strings. For more information see [Wildcards In Path](#).
- `access_key_id` and `secret_access_key` — Keys that specify credentials to use with given endpoint. Optional.

- `format` — The [format](#) of the file.
- `structure` — Structure of the table. Format '`column1_name column1_type, column2_name column2_type, ...`'.

## Returned value

A table with the specified structure for reading or writing data in the specified file.

## Examples

Select the data from all files in the cluster `cluster_simple`:

```
SELECT * FROM s3Cluster('cluster_simple', 'http://minio1:9001/root/data/{clickhouse,database}/*', 'minio', 'minio123', 'CSV', 'name String, value UInt32, polygon Array(Array(Tuple(Float64, Float64))))' ORDER BY (name, value, polygon);
```

Count the total amount of rows in all files in the cluster `cluster_simple`:

```
SELECT count(*) FROM s3Cluster('cluster_simple', 'http://minio1:9001/root/data/{clickhouse,database}/*', 'minio', 'minio123', 'CSV', 'name String, value UInt32, polygon Array(Array(Tuple(Float64, Float64))))');
```

## Warning

If your listing of files contains number ranges with leading zeros, use the construction with braces for each digit separately or use `?`

## See Also

- [S3 engine](#)
- [s3 table function](#)

## sqlite

Allows to perform queries on a data stored in an [SQLite](#) database.

## Syntax

```
sqlite('db_path', 'table_name')
```

## Arguments

- `db_path` — Path to a file with an SQLite database. [String](#).
- `table_name` — Name of a table in the SQLite database. [String](#).

## Returned value

- A table object with the same columns as in the original [SQLite](#) table.

## Example

Query:

```
SELECT * FROM sqlite('sqlite.db', 'table1') ORDER BY col2;
```

Result:

col1	col2
line1	1
line2	2
line3	3

## See Also

- [SQLite table engine](#)

# Dictionaries

A dictionary is a mapping (key -> attributes) that is convenient for various types of reference lists.

ClickHouse supports special functions for working with dictionaries that can be used in queries. It is easier and more efficient to use dictionaries with functions than a `JOIN` with reference tables.

ClickHouse supports:

- [Built-in dictionaries](#) with a specific [set of functions](#).
- [Plug-in \(external\) dictionaries](#) with a [set of functions](#).

# External Dictionaries

You can add your own dictionaries from various data sources. The data source for a dictionary can be a local text or executable file, an HTTP(s) resource, or another DBMS. For more information, see “[Sources for external dictionaries](#)”.

ClickHouse:

- Fully or partially stores dictionaries in RAM.
- Periodically updates dictionaries and dynamically loads missing values. In other words, dictionaries can be loaded dynamically.
- Allows to create external dictionaries with xml files or [DDL queries](#).

The configuration of external dictionaries can be located in one or more xml-files. The path to the configuration is specified in the [dictionaries\\_config](#) parameter.

Dictionaries can be loaded at server startup or at first use, depending on the [dictionaries\\_lazy\\_load](#) setting.

The [dictionaries](#) system table contains information about dictionaries configured at server. For each dictionary you can find there:

- Status of the dictionary.
- Configuration parameters.
- Metrics like amount of RAM allocated for the dictionary or a number of queries since the dictionary was successfully loaded.

The dictionary configuration file has the following format:

```

<clickhouse>
    <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>
</clickhouse>

```

You can [configure](#) any number of dictionaries in the same file.

[DDL queries for dictionaries](#) does not require any additional records in server configuration. They allow to work with dictionaries as first-class entities, like tables or views.

## Attention

You can convert values for a small dictionary by describing it in a `SELECT` query (see the [transform](#) function). This functionality is not related to external dictionaries.

## See Also

- [Configuring an External Dictionary](#)
- [Storing Dictionaries in Memory](#)
- [Dictionary Updates](#)
- [Sources of External Dictionaries](#)
- [Dictionary Key and Fields](#)
- [Functions for Working with External Dictionaries](#)

## Configuring an External Dictionary

If dictionary is configured using xml file, than dictionary configuration has the following structure:

```

<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>

```

Corresponding [DDL-query](#) has the following structure:

```
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\_\-].
- **source** — Source of the dictionary.
- **layout** — Dictionary layout in memory.
- **structure** — Structure of the dictionary . A key and attributes that can be retrieved by this key.
- **lifetime** — Frequency of dictionary updates.

## Storing Dictionaries in Memory

There are a variety of ways to store dictionaries in memory.

We recommend [flat](#), [hashed](#) and [complex\\_key\\_hashed](#), which provide optimal processing speed.

Caching is not recommended because of potentially poor performance and difficulties in selecting optimal parameters. Read more in the section [cache](#).

There are several ways to improve dictionary performance:

- Call the function for working with the dictionary after [GROUP BY](#).
- Mark attributes to extract as injective. An attribute is called injective if different attribute values correspond to different keys. So when [GROUP BY](#) uses a function that fetches an attribute value by the key, this function is automatically taken out of [GROUP BY](#).

ClickHouse generates an exception for errors with dictionaries. Examples of errors:

- The dictionary being accessed could not be loaded.
- Error querying a [cached](#) dictionary.

You can view the list of external dictionaries and their statuses in the `system.dictionaries` table.

The configuration looks like this:

```
<clickhouse>
    <dictionary>
        ...
        <layout>
            <layout_type>
                <!-- layout settings -->
            </layout_type>
        </layout>
        ...
    </dictionary>
</clickhouse>
```

Corresponding [DDL-query](#):

```
CREATE DICTIONARY (...)  
...  
LAYOUT(LAYOUT_TYPE(param value)) -- layout settings  
...
```

## Ways to Store Dictionaries in Memory

- flat
- hashed
- sparse\_hashed
- complex\_key\_hashed
- complex\_key\_sparse\_hashed
- hashed\_array
- complex\_key\_hashed\_array
- range\_hashed
- complex\_key\_range\_hashed
- cache
- complex\_key\_cache
- ssd\_cache
- complex\_key(ssd)\_cache
- direct
- complex\_key\_direct
- ip\_trie

### flat

The dictionary is completely stored in memory in the form of flat arrays. How much memory does the dictionary use? The amount is proportional to the size of the largest key (in space used).

The dictionary key has the `UInt64` type and the value is limited to `max_array_size` (by default — 500,000). If a larger key is discovered when creating the dictionary, ClickHouse throws an exception and does not create the dictionary. Dictionary flat arrays initial size is controlled by `initial_array_size` setting (by default — 1024).

All types of sources are supported. When updating, data (from a file or from a table) is read in its entirety.

This method provides the best performance among all available methods of storing the dictionary.

Configuration example:

```
<layout>  
  <flat>  
    <initial_array_size>50000</initial_array_size>  
    <max_array_size>5000000</max_array_size>  
  </flat>  
</layout>
```

or

```
LAYOUT(FLAT(INITIAL_ARRAY_SIZE 50000 MAX_ARRAY_SIZE 5000000))
```

## hashed

The dictionary is completely stored in memory in the form of a hash table. The dictionary can contain any number of elements with any identifiers In practice, the number of keys can reach tens of millions of items.

If `preallocate` is `true` (default is `false`) the hash table will be preallocated (this will make the dictionary load faster). But note that you should use it only if:

- The source support an approximate number of elements (for now it is supported only by the ClickHouse source).
- There are no duplicates in the data (otherwise it may increase memory usage for the hashtable).

All types of sources are supported. When updating, data (from a file or from a table) is read in its entirety.

Configuration example:

```
<layout>
  <hashed>
    <preallocate>0</preallocate>
  </hashed>
</layout>
```

or

```
LAYOUT(HASHED(PREALLOCATE 0))
```

## sparse\_hashed

Similar to `hashed`, but uses less memory in favor more CPU usage.

It will be also preallocated so as `hashed` (with `preallocate` set to `true`), and note that it is even more significant for `sparse_hashed`.

Configuration example:

```
<layout>
  <sparse_hashed />
</layout>
```

or

```
LAYOUT(SPARSE_HASHED([PREALLOCATE 0]))
```

## complex\_key\_hashed

This type of storage is for use with composite `keys`. Similar to `hashed`.

Configuration example:

```
<layout>
  <complex_key_hashed />
</layout>
```

or

```
LAYOUT(COMPLEX_KEY_HASHED())
```

## complex\_key\_sparse\_hashed

This type of storage is for use with composite [keys](#). Similar to [sparse\\_hashed](#).

Configuration example:

```
<layout>
<complex_key_sparse_hashed />
</layout>
```

or

```
LAYOUT(COMPLEX_KEY_SPARSE_HASHED())
```

## hashed\_array

The dictionary is completely stored in memory. Each attribute is stored in an array. The key attribute is stored in the form of a hashed table where value is an index in the attributes array. The dictionary can contain any number of elements with any identifiers. In practice, the number of keys can reach tens of millions of items.

All types of sources are supported. When updating, data (from a file or from a table) is read in its entirety.

Configuration example:

```
<layout>
<hashed_array>
</hashed_array>
</layout>
```

or

```
LAYOUT(HASHED_ARRAY())
```

## complex\_key\_hashed\_array

This type of storage is for use with composite [keys](#). Similar to [hashed\\_array](#).

Configuration example:

```
<layout>
<complex_key_hashed_array />
</layout>
```

or

```
LAYOUT(COMPLEX_KEY_HASHED_ARRAY())
```

## range\_hashed

The dictionary is stored in memory in the form of a hash table with an ordered array of ranges and their corresponding values.

This storage method works the same way as hashed and allows using date/time (arbitrary numeric type) ranges in addition to the key.

Example: The table contains discounts for each advertiser in the format:

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

To use a sample for date ranges, define the `range_min` and `range_max` elements in the [structure](#). These elements must contain elements `name` and `type` (if `type` is not specified, the default type will be used - Date). `type` can be any numeric type (Date / DateTime / UInt64 / Int32 / others).

## Warning

Values of `range_min` and `range_max` should fit in `Int64` type.

Example:

```
<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>
...
...
```

or

```
CREATE DICTIONARY somedict (
  id UInt64,
  first Date,
  last Date
)
PRIMARY KEY id
LAYOUT(RANGE_HASHED())
RANGE(MIN first MAX last)
```

To work with these dictionaries, you need to pass an additional argument to the `dictGetT` function, for which a range is selected:

```
dictGetT('dict_name', 'attr_name', id, date)
```

This function returns the value for the specified `ids` and the date range that includes the passed date.

Details of the algorithm:

- If the `id` is not found or a range is not found for the `id`, it returns the default value for the dictionary.
- If there are overlapping ranges, it returns value for any (random) range.
- If the range delimiter is `NULL` or an invalid date (such as `1900-01-01`), the range is open. The range can be open on both sides.

Configuration example:

```
<clickhouse>
  <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>
</clickhouse>
```

or

```
CREATE DICTIONARY somedict(
  Abcdef UInt64,
  StartTimeStamp UInt64,
  EndTimeStamp UInt64,
  XXXType String DEFAULT ""
)
PRIMARY KEY Abcdef
RANGE(MIN StartTimeStamp MAX EndTimeStamp)
```

## complex\_key\_range\_hashed

The dictionary is stored in memory in the form of a hash table with an ordered array of ranges and their corresponding values (see [range\\_hashed](#)). This type of storage is for use with composite [keys](#).

Configuration example:

```
CREATE DICTIONARY range_dictionary
(
    CountryID UInt64,
    CountryKey String,
    StartDate Date,
    EndDate Date,
    Tax Float64 DEFAULT 0.2
)
PRIMARY KEY CountryID, CountryKey
SOURCE(CLICKHOUSE(TABLE 'date_table'))
LIFETIME(MIN 1 MAX 1000)
LAYOUT(COMPLEX_KEY_RANGE_HASHED())
RANGE(MIN StartDate MAX EndDate);
```

## cache

The dictionary is stored in a cache that has a fixed number of cells. These cells contain frequently used elements.

When searching for a dictionary, the cache is searched first. For each block of data, all keys that are not found in the cache or are outdated are requested from the source using `SELECT attrs... FROM db.table WHERE id IN (k1, k2, ...)`. The received data is then written to the cache.

If keys are not found in dictionary, then update cache task is created and added into update queue. Update queue properties can be controlled with settings `max_update_queue_size`,  
`update_queue_push_timeout_milliseconds`, `query_wait_timeout_milliseconds`, `max_threads_for_updates`.

For cache dictionaries, the expiration `lifetime` of data in the cache can be set. If more time than lifetime has passed since loading the data in a cell, the cell's value is not used and key becomes expired. The key is re-requested the next time it needs to be used. This behaviour can be configured with setting `allow_read_expired_keys`.

This is the least effective of all the ways to store dictionaries. The speed of the cache depends strongly on correct settings and the usage scenario. A cache type dictionary performs well only when the hit rates are high enough (recommended 99% and higher). You can view the average hit rate in the `system.dictionaries` table.

If setting `allow_read_expired_keys` is set to 1, by default 0. Then dictionary can support asynchronous updates. If a client requests keys and all of them are in cache, but some of them are expired, then dictionary will return expired keys for a client and request them asynchronously from the source.

To improve cache performance, use a subquery with `LIMIT`, and call the function with the dictionary externally.

Supported `sources`: MySQL, ClickHouse, executable, HTTP.

Example of settings:

```

<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>
    <!-- Allows to read expired keys. -->
    <allow_read_expired_keys>0</allow_read_expired_keys>
    <!-- Max size of update queue. -->
    <max_update_queue_size>100000</max_update_queue_size>
    <!-- Max timeout in milliseconds for push update task into queue. -->
    <update_queue_push_timeout_milliseconds>10</update_queue_push_timeout_milliseconds>
    <!-- Max wait timeout in milliseconds for update task to complete. -->
    <query_wait_timeout_milliseconds>60000</query_wait_timeout_milliseconds>
    <!-- Max threads for cache dictionary update. -->
    <max_threads_for_updates>4</max_threads_for_updates>
  </cache>
</layout>

```

or

```
LAYOUT(CACHE(SIZE_IN_CELLS 1000000000))
```

Set a large enough cache size. You need to experiment to select the number of cells:

1. Set some value.
2. Run queries until the cache is completely full.
3. Assess memory consumption using the `system.dictionaries` table.
4. Increase or decrease the number of cells until the required memory consumption is reached.

## Warning

Do not use ClickHouse as a source, because it is slow to process queries with random reads.

## complex\_key\_cache

This type of storage is for use with composite [keys](#). Similar to `cache`.

## ssd\_cache

Similar to `cache`, but stores data on SSD and index in RAM. All cache dictionary settings related to update queue can also be applied to SSD cache dictionaries.

```

<layout>
  <ssd_cache>
    <!-- Size of elementary read block in bytes. Recommended to be equal to SSD's page size. -->
    <block_size>4096</block_size>
    <!-- Max cache file size in bytes. -->
    <file_size>16777216</file_size>
    <!-- Size of RAM buffer in bytes for reading elements from SSD. -->
    <read_buffer_size>131072</read_buffer_size>
    <!-- Size of RAM buffer in bytes for aggregating elements before flushing to SSD. -->
    <write_buffer_size>1048576</write_buffer_size>
    <!-- Path where cache file will be stored. -->
    <path>/var/lib/clickhouse/user_files/test_dict</path>
  </ssd_cache>
</layout>

```

or

```
LAYOUT(SSD_CACHE(BLOCK_SIZE 4096 FILE_SIZE 16777216 READ_BUFFER_SIZE 1048576
  PATH '/var/lib/clickhouse/user_files/test_dict'))
```

## **complex\_key\_ssd\_cache**

This type of storage is for use with composite **keys**. Similar to **ssd\_cache**.

### **direct**

The dictionary is not stored in memory and directly goes to the source during the processing of a request.

The dictionary key has the **UInt64** type.

All types of **sources**, except local files, are supported.

Configuration example:

```
<layout>
<direct />
</layout>
```

or

```
LAYOUT(DIRECT())
```

## **complex\_key\_direct**

This type of storage is for use with composite **keys**. Similar to **direct**.

### **ip\_trie**

This type of storage is for mapping network prefixes (IP addresses) to metadata such as ASN.

Example: The table contains network prefixes and their corresponding AS number and country code:

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

When using this type of layout, the structure must have a composite key.

Example:

```

<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>
  ...
</structure>
<layout>
  <ip_trie>
    <!-- Key attribute `prefix` can be retrieved via dictGetString. -->
    <!-- This option increases memory usage. -->
    <access_to_key_from_attributes>true</access_to_key_from_attributes>
  </ip_trie>
</layout>

```

or

```

CREATE DICTIONARY somedict (
  prefix String,
  asn UInt32,
  cca2 String DEFAULT '??'
)
PRIMARY KEY prefix

```

The key must have only one String type attribute that contains an allowed IP prefix. Other types are not supported yet.

For queries, you must use the same functions (`dictGetT` with a tuple) as for dictionaries with composite keys:

```
dictGetT('dict_name', 'attr_name', tuple(ip))
```

The function takes either `UInt32` for IPv4, or `FixedString(16)` for IPv6:

```
dictGetString('prefix', 'asn', tuple(IPv6StringToNum('2001:db8::1')))
```

Other types are not supported yet. The function returns the attribute for the prefix that corresponds to this IP address. If there are overlapping prefixes, the most specific one is returned.

Data must completely fit into RAM.

## Dictionary Updates

ClickHouse periodically updates the dictionaries. The update interval for fully downloaded dictionaries and the invalidation interval for cached dictionaries are defined in the `<lifetime>` tag in seconds.

Dictionary updates (other than loading for first use) do not block queries. During updates, the old version of a dictionary is used. If an error occurs during an update, the error is written to the server log, and queries continue using the old version of dictionaries.

Example of settings:

```
<dictionary>
...
<lifetime>300</lifetime>
...
</dictionary>
```

or

```
CREATE DICTIONARY (...)

...
LIFETIME(300)
...
```

Setting `<lifetime>0</lifetime>` (`LIFETIME(0)`) prevents dictionaries from updating.

You can set a time interval for updates, and ClickHouse will choose a uniformly random time within this range. This is necessary in order to distribute the load on the dictionary source when updating on a large number of servers.

Example of settings:

```
<dictionary>
...
<lifetime>
  <min>300</min>
  <max>360</max>
</lifetime>
...
</dictionary>
```

or

```
LIFETIME(MIN 300 MAX 360)
```

If `<min>0</min>` and `<max>0</max>`, ClickHouse does not reload the dictionary by timeout.

In this case, ClickHouse can reload the dictionary earlier if the dictionary configuration file was changed or the `SYSTEM RELOAD DICTIONARY` command was executed.

When updating the dictionaries, the ClickHouse server applies different logic depending on the type of [source](#):

- For a text file, it checks the time of modification. If the time differs from the previously recorded time, the dictionary is updated.
- For MySQL source, the time of modification is checked using a `SHOW TABLE STATUS` query (in case of MySQL 8 you need to disable meta-information caching in MySQL by `set global information_schema_stats_expiry=0`).
- Dictionaries from other sources are updated every time by default.

For other sources (ODBC, PostgreSQL, ClickHouse, etc), you can set up a query that will update the dictionaries only if they really changed, rather than each time. To do this, follow these steps:

- The dictionary table must have a field that always changes when the source data is updated.

- The settings of the source must specify a query that retrieves the changing field. The ClickHouse server interprets the query result as a row, and if this row has changed relative to its previous state, the dictionary is updated. Specify the query in the `<invalidate_query>` field in the settings for the [source](#).

Example of settings:

```
<dictionary>
...
<odbc>
...
  <invalidate_query>SELECT update_time FROM dictionary_source where id = 1</invalidate_query>
</odbc>
...
</dictionary>
```

or

```
...
SOURCE(ODBC(... invalidate_query 'SELECT update_time FROM dictionary_source where id = 1')
...
...
```

For `Cache`, `ComplexKeyCache`, `SSDCache`, and `SSDComplexKeyCache` dictionaries both synchronious and asynchronous updates are supported.

It is also possible for `Flat`, `Hashed`, `ComplexKeyHashed` dictionaries to only request data that was changed after the previous update. If `update_field` is specified as part of the dictionary source configuration, value of the previous update time in seconds will be added to the data request. Depends on source type (`Executable`, `HTTP`, `MySQL`, `PostgreSQL`, `ClickHouse`, or `ODBC`) different logic will be applied to `update_field` before request data from an external source.

- If the source is `HTTP` then `update_field` will be added as a query parameter with the last update time as the parameter value.
- If the source is `Executable` then `update_field` will be added as an executable script argument with the last update time as the argument value.
- If the source is `ClickHouse`, `MySQL`, `PostgreSQL`, `ODBC` there will be an additional part of `WHERE`, where `update_field` is compared as greater or equal with the last update time.

If `update_field` option is set, additional option `update_lag` can be set. Value of `update_lag` option is subtracted from previous update time before request updated data.

Example of settings:

```
<dictionary>
...
<clickhouse>
...
  <update_field>added_time</update_field>
  <update_lag>15</update_lag>
</clickhouse>
...
</dictionary>
```

or

```
...
SOURCE(CLICKHOUSE(... update_field 'added_time' update_lag 15))
...
...
```

# Sources of External Dictionaries

An external dictionary can be connected from many different sources.

If dictionary is configured using xml-file, the configuration looks like this:

```
<clickhouse>
<dictionary>
...
<source>
<source_type>
    <!-- Source configuration -->
</source_type>
</source>
...
</dictionary>
...
</clickhouse>
```

In case of [DDL-query](#), equal configuration will looks like:

```
CREATE DICTIONARY dict_name (...)

...
SOURCE(SOURCE_TYPE(param1 val1 ... paramN valN)) -- Source configuration
...
```

The source is configured in the source section.

For source types [Local file](#), [Executable file](#), [HTTP\(s\)](#), [ClickHouse](#) optional settings are available:

```
<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>
```

or

```
SOURCE(FILE(path './user_files/os.tsv' format 'TabSeparated'))
SETTINGS(format_csv_allow_single_quotes = 0)
```

Types of sources (source\_type):

- [Local file](#)
- [Executable File](#)
- [Executable Pool](#)
- [HTTP\(s\)](#)

- DBMS
  - ODBC
  - MySQL
  - PostgreSQL
  - ClickHouse
  - MongoDB
  - Redis
  - Cassandra
  - PostgreSQL

## Local File

Example of settings:

```
<source>
  <file>
    <path>/opt/dictionaries/os.tsv</path>
    <format>TabSeparated</format>
  </file>
</source>
```

or

```
SOURCE(FILE(path './user_files/os.tsv' format 'TabSeparated'))
```

Setting fields:

- `path` – The absolute path to the file.
- `format` – The file format. All the formats described in [Formats](#) are supported.

When dictionary with source `FILE` is created via DDL command (`CREATE DICTIONARY ...`), the source file needs to be located in `user_files` directory, to prevent DB users accessing arbitrary file on ClickHouse node.

## See Also

- [Dictionary function](#)

## Executable File

Working with executable files depends on [how the dictionary is stored in memory](#). If the dictionary is stored using `cache` and `complex_key_cache`, ClickHouse requests the necessary keys by sending a request to the executable file's STDIN. Otherwise, ClickHouse starts executable file and treats its output as dictionary data.

Example of settings:

```
<source>
  <executable>
    <command>cat /opt/dictionaries/os.tsv</command>
    <format>TabSeparated</format>
    <implicit_key>false</implicit_key>
  </executable>
</source>
```

Setting fields:

- `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 [Formats](#) are supported.
- `implicit_key` — The executable source file can return only values, and the correspondence to the requested keys is determined implicitly — by the order of rows in the result. Default value is false.

That dictionary source can be configured only via XML configuration. Creating dictionaries with executable source via DDL is disabled, otherwise, the DB user would be able to execute arbitrary binary on ClickHouse node.

## Executable Pool

Executable pool allows loading data from pool of processes. This source does not work with dictionary layouts that need to load all data from source. Executable pool works if the dictionary is stored using `cache`, `complex_key_cache`, `ssd_cache`, `complex_key_ssd_cache`, `direct`, `complex_key_direct` layouts.

Executable pool will spawn pool of processes with specified command and keep them running until they exit. The program should read data from STDIN while it is available and output result to STDOUT, and it can wait for next block of data on STDIN. ClickHouse will not close STDIN after processing a block of data but will pipe another chunk of data when needed. The executable script should be ready for this way of data processing — it should poll STDIN and flush data to STDOUT early.

Example of settings:

```
<source>
  <executable_pool>
    <command><command>while read key; do printf "$key\tData for key $key\n"; done</command></command>
    <format>TabSeparated</format>
    <pool_size>10</pool_size>
    <max_command_execution_time>10</max_command_execution_time>
    <implicit_key>false</implicit_key>
  </executable_pool>
</source>
```

Setting fields:

- `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 “[Formats](#)” are supported.
- `pool_size` — Size of pool. If 0 is specified as `pool_size` then there is no pool size restrictions.
- `command_termination_timeout` — Executable pool script should contain main read-write loop. After dictionary is destroyed, pipe is closed, and executable file will have `command_termination_timeout` seconds to shutdown, before ClickHouse will send SIGTERM signal to child process. Specified in seconds. Default value is 10. Optional parameter.
- `max_command_execution_time` — Maximum executable script command execution time for processing block of data. Specified in seconds. Default value is 10. Optional parameter.
- `implicit_key` — The executable source file can return only values, and the correspondence to the requested keys is determined implicitly — by the order of rows in the result. Default value is false. Optional parameter.

That dictionary source can be configured only via XML configuration. Creating dictionaries with executable source via DDL is disabled, otherwise, the DB user would be able to execute arbitrary binary on ClickHouse node.

## Http(s)

Working with an HTTP(s) server depends on [how the dictionary is stored in memory](#). If the dictionary is stored using `cache` and `complex_key_cache`, ClickHouse requests the necessary keys by sending a request via the `POST` method.

Example of settings:

```
<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>
```

or

```
SOURCE(HTTP(
  url 'http://[::1]/os.tsv'
  format 'TabSeparated'
  credentials(user 'user' password 'password')
  headers(header(name 'API-KEY' value 'key'))
))
```

In order for ClickHouse to access an HTTPS resource, you must [configure openSSL](#) in the server configuration.

Setting fields:

- `url` – The source URL.
- `format` – The file format. All the formats described in “[Formats](#)” are supported.
- `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.

When creating a dictionary using the DDL command (CREATE DICTIONARY ...) remote hosts for HTTP dictionaries are checked against the contents of `remote_url_allow_hosts` section from config to prevent database users to access arbitrary HTTP server.

## ODBC

You can use this method to connect any database that has an ODBC driver.

Example of settings:

```
<source>
  <odbc>
    <db>DatabaseName</db>
    <table>SchemaName.TableName</table>
    <connection_string>DSN=some_parameters</connection_string>
    <invalidate_query>SQL_QUERY</invalidate_query>
  </odbc>
</source>
```

or

```
SOURCE(ODBC(
  db 'DatabaseName'
  table 'SchemaName.TableName'
  connection_string 'DSN=some_parameters'
  invalidate_query 'SQL_QUERY'
))
```

Setting fields:

- `db` – Name of the database. Omit it if the database name is set in the `<connection_string>` parameters.
- `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 [Updating dictionaries](#).

ClickHouse receives quoting symbols from ODBC-driver and quote all settings in queries to driver, so it's necessary to set table name accordingly to table name case in database.

If you have a problems with encodings when using Oracle, see the corresponding [F.A.Q.](#) item.

## Known Vulnerability of the ODBC Dictionary Functionality

### Attention

When connecting to the database through the ODBC driver connection parameter `Servername` can be substituted. In this case values of `USERNAME` and `PASSWORD` from `odbc.ini` are sent to the remote server and can be compromised.

### Example of insecure use

Let's configure unixODBC for PostgreSQL. Content of `/etc/odbc.ini`:

```
[gregtest]
Driver = /usr/lib/psqlodbc.so
Servername = localhost
PORT = 5432
DATABASE = test_db
##OPTION = 3
USERNAME = test
PASSWORD = test
```

If you then make a query such as

```
SELECT * FROM odbc('DSN=gregtest;Servername=some-server.com', 'test_db');
```

ODBC driver will send values of `USERNAME` and `PASSWORD` from `odbc.ini` to `some-server.com`.

## Example of Connecting Postgresql

Ubuntu OS.

Installing unixODBC and the ODBC driver for PostgreSQL:

```
$ sudo apt-get install -y unixodbc odbcinst odbc-postgresql
```

Configuring `/etc/odbc.ini` (or `~/.odbc.ini` if you signed in under a user that runs ClickHouse):

```
[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    =
```

The dictionary configuration in ClickHouse:

```

<clickhouse>
  <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>
</clickhouse>

```

or

```

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)

```

You may need to edit `odbc.ini` to specify the full path to the library with the driver `DRIVER=/usr/local/lib/psqlodbcw.so`.

## Example of Connecting MS SQL Server

Ubuntu OS.

Installing the ODBC driver for connecting to MS SQL:

```
$ sudo apt-get install tdsodbc freetds-bin sqsh
```

Configuring the driver:

```

$ cat /etc/freetds/freetds.conf
...
[MSSQL]
host = 192.168.56.101
port = 1433
tds version = 7.0
client charset = UTF-8

# test TDS connection
$ sqsh -S MSSQL -D database -U user -P password

$ 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 /etc/odbc.ini
# $ cat ~/.odbc.ini # if you signed in under a user that runs ClickHouse

[MSSQL]
Description = FreeTDS
Driver = FreeTDS
Servername = MSSQL
Database = test
UID = test
PWD = test
Port = 1433

# (optional) test ODBC connection (to use isql-tool install the [unixodbc](https://packages.debian.org/sid/unixodbc)-package)
$ isql -v MSSQL "user" "password"

```

#### Remarks:

- to determine the earliest TDS version that is supported by a particular SQL Server version, refer to the product documentation or look at [MS-TDS Product Behavior](#)

Configuring the dictionary in ClickHouse:

```

<clickhouse>
  <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>
</clickhouse>

```

or

```

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

Example of settings:

```

<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>
    <replica>
      <host>example01-3</host>
      <priority>1</priority>
    </replica>
    <db>db_name</db>
    <table>table_name</table>
    <where>id=10</where>
    <invalidate_query>SQL_QUERY</invalidate_query>
    <fail_on_connection_loss>true</fail_on_connection_loss>
  </mysql>
</source>

```

or

```
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'
    fail_on_connection_loss 'true'
))
```

Setting fields:

- **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** clause in MySQL, for example, `id > 10 AND id < 20`. Optional parameter.
- **invalidate\_query** – Query for checking the dictionary status. Optional parameter. Read more in the section [Updating dictionaries](#).
- **fail\_on\_connection\_loss** – The configuration parameter that controls behavior of the server on connection loss. If `true`, an exception is thrown immediately if the connection between client and server was lost. If `false`, the ClickHouse server retries to execute the query three times before throwing an exception. Note that retrying leads to increased response times. Default value: `false`.

MySQL can be connected on a local host via sockets. To do this, set `host` and `socket`.

Example of settings:

```

<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>
    <fail_on_connection_loss>true</fail_on_connection_loss>
  </mysql>
</source>

```

or

```

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'
  fail_on_connection_loss 'true'
))

```

## ClickHouse

Example of settings:

```

<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>
    <secure>1</secure>
  </clickhouse>
</source>

```

or

```

SOURCE(CLICKHOUSE(
  host 'example01-01-1'
  port 9000
  user 'default'
  password ""
  db 'default'
  table 'ids'
  where 'id=10'
  secure 1
));

```

Setting fields:

- **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 **Distributed** table and enter it in subsequent configurations.
- **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 [Updating dictionaries](#).
- `secure` - Use ssl for connection.

## Mongodb

Example of settings:

```
<source>
  <mongodb>
    <host>localhost</host>
    <port>27017</port>
    <user></user>
    <password></password>
    <db>test</db>
    <collection>dictionary_source</collection>
  </mongodb>
</source>
```

or

```
SOURCE(MONGODB(
  host 'localhost'
  port 27017
  user ""
  password ""
  db 'test'
  collection 'dictionary_source'
))
```

Setting fields:

- `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

Example of settings:

```
<source>
  <redis>
    <host>localhost</host>
    <port>6379</port>
    <storage_type>simple</storage_type>
    <db_index>0</db_index>
  </redis>
</source>
```

or

```
SOURCE(REDIS(
  host 'localhost'
  port 6379
  storage_type 'simple'
  db_index 0
))
```

Setting fields:

- **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** is for simple sources and for hashed single key sources, **hash\_map** is for hashed sources with two keys. Ranged sources and cache sources with complex key are unsupported. May be omitted, default value is **simple**.
- **db\_index** – The specific numeric index of Redis logical database. May be omitted, default value is 0.

## Cassandra

Example of settings:

```
<source>
  <cassandra>
    <host>localhost</host>
    <port>9042</port>
    <user>username</user>
    <password>qwerty123</password>
    <keyspase>database_name</keyspase>
    <column_family>table_name</column_family>
    <allow_filering>1</allow_filering>
    <partition_key_prefix>1</partition_key_prefix>
    <consistency>One</consistency>
    <where>"SomeColumn" = 42</where>
    <max_threads>8</max_threads>
  </cassandra>
</source>
```

#### Setting fields:

- `host` – The Cassandra host or comma-separated list of hosts.
- `port` – The port on the Cassandra servers. If not specified, default port 9042 is used.
- `user` – Name of the Cassandra user.
- `password` – Password of the Cassandra user.
- `keyspace` – Name of the keyspace (database).
- `column_family` – Name of the column family (table).
- `allow_filerig` – Flag to allow or not potentially expensive conditions on clustering key columns. Default value is 1.
- `partition_key_prefix` – Number of partition key columns in primary key of the Cassandra table.

Required for compose key dictionaries. Order of key columns in the dictionary definition must be the same as in Cassandra.

Default value is 1 (the first key column is a partition key and other key columns are clustering key).

- `consistency` – Consistency level. Possible values: `One`, `Two`, `Three`, `All`, `EachQuorum`, `Quorum`, `LocalQuorum`, `LocalOne`, `Serial`, `LocalSerial`. Default is `One`.
- `where` – Optional selection criteria.
- `max_threads` – The maximum number of threads to use for loading data from multiple partitions in compose key dictionaries.

## PostgreSQL

#### Example of settings:

```
<source>
<postgresql>
  <port>5432</port>
  <user>clickhouse</user>
  <password>qwert</password>
  <db>db_name</db>
  <table>table_name</table>
  <where>id=10</where>
  <invalidate_query>SQL_QUERY</invalidate_query>
</postgresql>
</source>
```

or

```
SOURCE(POSTGRESQL(
  port 5432
  host 'postgresql-hostname'
  user 'postgres_user'
  password 'postgres_password'
  db 'db_name'
  table 'table_name'
  replica(host 'example01-1' port 5432 priority 1)
  replica(host 'example01-2' port 5432 priority 2)
  where 'id=10'
  invalidate_query 'SQL_QUERY'
))
```

#### Setting fields:

- `host` – The host on the PostgreSQL server. You can specify it for all replicas, or for each one individually (inside `<replica>`).
- `port` – The port on the PostgreSQL server. You can specify it for all replicas, or for each one individually (inside `<replica>`).
- `user` – Name of the PostgreSQL user. You can specify it for all replicas, or for each one individually (inside `<replica>`).

- `password` – Password of the PostgreSQL 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 PostgreSQL host.
  - `replica/port` – The PostgreSQL port.
  - `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` clause in PostgreSQL, for example, `id > 10 AND id < 20`. Optional parameter.
- `invalidate_query` – Query for checking the dictionary status. Optional parameter. Read more in the section [Updating dictionaries](#).

## Dictionary Key and Fields

The `<structure>` clause describes the dictionary key and fields available for queries.

XML description:

```

<dictionary>
  <structure>
    <id>
      <name>Id</name>
    </id>

    <attribute>
      <!-- Attribute parameters -->
    </attribute>

    ...
  </structure>
</dictionary>

```

Attributes are described in the elements:

- `<id>` — [Key column](#).
- `<attribute>` — [Data column](#). There can be a multiple number of attributes.

DDL query:

```

CREATE DICTIONARY dict_name (
  Id UInt64,
  -- attributes
)
PRIMARY KEY Id
...

```

Attributes are described in the query body:

- `PRIMARY KEY` — [Key column](#)
- `AttrName AttrType` — [Data column](#). There can be a multiple number of attributes.

# Key

ClickHouse supports the following types of keys:

- Numeric key. `UInt64`. Defined in the `<id>` tag or using `PRIMARY KEY` keyword.
- Composite key. Set of values of different types. Defined in the tag `<key>` or `PRIMARY KEY` keyword.

An xml structure can contain either `<id>` or `<key>`. DDL-query must contain single `PRIMARY KEY`.

## Warning

You must not describe key as an attribute.

## Numeric Key

Type: `UInt64`.

Configuration example:

```
<id>
  <name>Id</name>
</id>
```

Configuration fields:

- `name` – The name of the column with keys.

For DDL-query:

```
CREATE DICTIONARY (
  Id UInt64,
  ...
)
PRIMARY KEY Id
...
```

- `PRIMARY KEY` – The name of the column with keys.

## Composite Key

The key can be a `tuple` from any types of fields. The `layout` in this case must be `complex_key_hashed` or `complex_key_cache`.

## Tip

A composite key can consist of a single element. This makes it possible to use a string as the key, for instance.

The key structure is set in the element `<key>`. Key fields are specified in the same format as the dictionary [attributes](#). Example:

```

<structure>
  <key>
    <attribute>
      <name>field1</name>
      <type>String</type>
    </attribute>
    <attribute>
      <name>field2</name>
      <type>UInt32</type>
    </attribute>
  ...
</key>
...

```

or

```

CREATE DICTIONARY (
  field1 String,
  field2 String
  ...
)
PRIMARY KEY field1, field2
...

```

For a query to the `dictGet*` function, a tuple is passed as the key. Example: `dictGetString('dict_name', 'attr_name', tuple('string for field1', num_for_field2))`.

## Attributes

Configuration example:

```

<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>

```

or

```

CREATE DICTIONARY somename (
  Name ClickHouseDataType DEFAULT '' EXPRESSION rand64() HIERARCHICAL INJECTIVE IS_OBJECT_ID
)

```

Configuration fields:

Tag	Description	Required
name	Column name.	Yes

Tag	Description	Required
type	<p>ClickHouse data type: <code>UInt8</code>, <code>UInt16</code>, <code>UInt32</code>, <code>UInt64</code>, <code>Int8</code>, <code>Int16</code>, <code>Int32</code>, <code>Int64</code>, <code>Float32</code>, <code>Float64</code>, <code>UUID</code>, <code>Decimal32</code>, <code>Decimal64</code>, <code>Decimal128</code>, <code>Decimal256</code>, <code>String</code>, <code>Array</code>.</p> <p>ClickHouse tries to cast value from dictionary to the specified data type. For example, for MySQL, the field might be <code>TEXT</code>, <code>VARCHAR</code>, or <code>BLOB</code> in the MySQL source table, but it can be uploaded as <code>String</code> in ClickHouse.</p> <p><code>Nullable</code> is currently supported for <code>Flat</code>, <code>Hashed</code>, <code>ComplexKeyHashed</code>, <code>Direct</code>, <code>ComplexKeyDirect</code>, <code>RangeHashed</code>, <code>Polygon</code>, <code>Cache</code>, <code>ComplexKeyCache</code>, <code>SSDCache</code>, <code>SSDComplexKeyCache</code> dictionaries. In <code>IPTrie</code> dictionaries <code>Nullable</code> types are not supported.</p>	Yes
null_value	<p>Default value for a non-existing element.</p> <p>In the example, it is an empty string. <code>NULL</code> value can be used only for the <code>Nullable</code> types (see the previous line with types description).</p>	Yes
expression	<p><code>Expression</code> that ClickHouse executes on the value.</p> <p>The expression can be a column name in the remote SQL database. Thus, you can use it to create an alias for the remote column.</p> <p>Default value: no expression.</p>	No
hierarchical	<p>If <code>true</code>, the attribute contains the value of a parent key for the current key. See <a href="#">Hierarchical Dictionaries</a>.</p> <p>Default value: <code>false</code>.</p>	No
injective	<p>Flag that shows whether the <code>id -&gt; attribute</code> image is <code>injective</code>. If <code>true</code>, ClickHouse can automatically place after the <code>GROUP BY</code> clause the requests to dictionaries with injection. Usually it significantly reduces the amount of such requests.</p> <p>Default value: <code>false</code>.</p>	No
is_object_id	<p>Flag that shows whether the query is executed for a MongoDB document by <code>ObjectID</code>.</p> <p>Default value: <code>false</code>.</p>	No

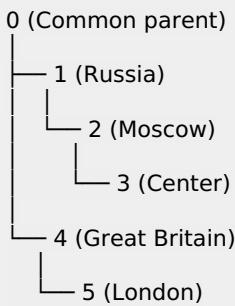
## See Also

- [Functions for working with external dictionaries](#).

## Hierarchical Dictionaries

ClickHouse supports hierarchical dictionaries with a `numeric key`.

Look at the following hierarchical structure:



This hierarchy can be expressed as the following dictionary table.

region_id	parent_region	region_name
1	0	Russia
2	1	Moscow
3	2	Center
4	0	Great Britain
5	4	London

This table contains a column `parent_region` that contains the key of the nearest parent for the element.

ClickHouse supports the `hierarchical` property for `external dictionary` attributes. This property allows you to configure the hierarchical dictionary similar to described above.

The `dictGetHierarchy` function allows you to get the parent chain of an element.

For our example, the structure of dictionary can be the following:

```

<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>
  
```

## Polygon dictionaries

Polygon dictionaries allow you to efficiently search for the polygon containing specified points. For example: defining a city area by geographical coordinates.

Example configuration:

```
<dictionary>
  <structure>
    <key>
      <name>key</name>
      <type>Array(Array(Array(Array(Float64))))</type>
    </key>

    <attribute>
      <name>name</name>
      <type>String</type>
      <null_value></null_value>
    </attribute>

    <attribute>
      <name>value</name>
      <type>UInt64</type>
      <null_value>0</null_value>
    </attribute>

  </structure>

  <layout>
    <polygon />
  </layout>
</dictionary>
```

The corresponding [DDL-query](#):

```
CREATE DICTIONARY polygon_dict_name (
  key Array(Array(Array(Array(Float64)))),
  name String,
  value UInt64
)
PRIMARY KEY key
LAYOUT(POLYGON())
...
```

When configuring the polygon dictionary, the key must have one of two types:

- A simple polygon. It is an array of points.
- MultiPolygon. It is an array of polygons. Each polygon is a two-dimensional array of points. The first element of this array is the outer boundary of the polygon, and subsequent elements specify areas to be excluded from it.

Points can be specified as an array or a tuple of their coordinates. In the current implementation, only two-dimensional points are supported.

The user can [upload their own data](#) in all formats supported by ClickHouse.

There are 3 types of [in-memory storage](#) available:

- POLYGON\_SIMPLE. This is a naive implementation, where a linear pass through all polygons is made for each query, and membership is checked for each one without using additional indexes.

- POLYGON\_INDEX\_EACH. A separate index is built for each polygon, which allows you to quickly check whether it belongs in most cases (optimized for geographical regions).  
Also, a grid is superimposed on the area under consideration, which significantly narrows the number of polygons under consideration.  
The grid is created by recursively dividing the cell into 16 equal parts and is configured with two parameters.  
The division stops when the recursion depth reaches MAX\_DEPTH or when the cell crosses no more than MIN\_INTERSECTIONS polygons.  
To respond to the query, there is a corresponding cell, and the index for the polygons stored in it is accessed alternately.
- POLYGON\_INDEX\_CELL. This placement also creates the grid described above. The same options are available. For each sheet cell, an index is built on all pieces of polygons that fall into it, which allows you to quickly respond to a request.
- POLYGON. Synonym to POLYGON\_INDEX\_CELL.

Dictionary queries are carried out using standard [functions](#) for working with external dictionaries. An important difference is that here the keys will be the points for which you want to find the polygon containing them.

Example of working with the dictionary defined above:

```
CREATE TABLE points (
    x Float64,
    y Float64
)
...
SELECT tuple(x, y) AS key, dictGet(dict_name, 'name', key), dictGet(dict_name, 'value', key) FROM points ORDER BY x, y;
```

As a result of executing the last command for each point in the 'points' table, a minimum area polygon containing this point will be found, and the requested attributes will be output.

## Internal Dictionaries

ClickHouse contains a built-in feature for working with a geobase.

This allows you to:

- Use a region's ID to get its name in the desired language.
- Use a region's ID to get the ID of a city, area, federal district, country, or continent.
- Check whether a region is part of another region.
- Get a chain of parent regions.

All the functions support “translocality,” the ability to simultaneously use different perspectives on region ownership. For more information, see the section “[Functions for working with Yandex.Metrica dictionaries](#)”.

The internal dictionaries are disabled in the default package.

To enable them, uncomment the parameters `path_to_regions_hierarchy_file` and `path_to_regions_names_files` in the server configuration file.

The geobase is loaded from text files.

Place the `regions_hierarchy*.txt` files into the `path_to_regions_hierarchy_file` directory. This configuration parameter must contain the path to the `regions_hierarchy.txt` file (the default regional hierarchy), and the other files (`regions_hierarchy_ua.txt`) must be located in the same directory.

Put the `regions_names_*.txt` files in the `path_to_regions_names_files` directory.

You can also create these files yourself. The file format is as follows:

`regions_hierarchy*.txt`: TabSeparated (no header), columns:

- region ID (`UInt32`)
- parent region ID (`UInt32`)
- region type (`UInt8`): 1 - continent, 3 - country, 4 - federal district, 5 - region, 6 - city; other types do not have values
- population (`UInt32`) — optional column

`regions_names_*.txt`: TabSeparated (no header), columns:

- region ID (`UInt32`)
- region name (`String`) — Can't contain tabs or line feeds, even escaped ones.

A flat array is used for storing in RAM. For this reason, IDs shouldn't be more than a million.

Dictionaries can be updated without restarting the server. However, the set of available dictionaries is not updated.

For updates, the file modification times are checked. If a file has changed, the dictionary is updated.

The interval to check for changes is configured in the `builtin_dictionaries_reload_interval` parameter.

Dictionary updates (other than loading at first use) do not block queries. During updates, queries use the old versions of dictionaries. If an error occurs during an update, the error is written to the server log, and queries continue using the old version of dictionaries.

We recommend periodically updating the dictionaries with the geobase. During an update, generate new files and write them to a separate location. When everything is ready, rename them to the files used by the server.

There are also functions for working with OS identifiers and Yandex.Metrica search engines, but they shouldn't be used.

---

## Data Types

ClickHouse can store various kinds of data in table cells.

This section describes the supported data types and special considerations for using and/or implementing them if any.

You can check whether data type name is case-sensitive in the `system.data_type_families` table.

---

## `UInt8`, `UInt16`, `UInt32`, `UInt64`, `UInt256`, `Int8`, `Int16`, `Int32`, `Int64`, `Int128`, `Int256`

Fixed-length integers, with or without a sign.

When creating tables, numeric parameters for integer numbers can be set (e.g. `TINYINT(8)`, `SMALLINT(16)`, `INT(32)`, `BIGINT(64)`), but ClickHouse ignores them.

## Int Ranges

- `Int8` — [-128 : 127]
- `Int16` — [-32768 : 32767]
- `Int32` — [-2147483648 : 2147483647]
- `Int64` — [-9223372036854775808 : 9223372036854775807]
- `Int128` — [-170141183460469231731687303715884105728 : 170141183460469231731687303715884105727]
- `Int256` — [-57896044618658097711785492504343953926634992332820282019728792003956564819968 : 57896044618658097711785492504343953926634992332820282019728792003956564819967]

Aliases:

- `Int8` — `TINYINT`, `BOOL`, `BOOLEAN`, `INT1`.
- `Int16` — `SMALLINT`, `INT2`.
- `Int32` — `INT`, `INT4`, `INTEGER`.
- `Int64` — `BIGINT`.

## UInt Ranges

- `UInt8` — [0 : 255]
- `UInt16` — [0 : 65535]
- `UInt32` — [0 : 4294967295]
- `UInt64` — [0 : 18446744073709551615]
- `UInt128` — [0 : 340282366920938463463374607431768211455]
- `UInt256` — [0 : 115792089237316195423570985008687907853269984665640564039457584007913129639935]

## Float32, Float64

Floating point numbers.

Types are equivalent to types of C:

- `Float32` — `float`.
- `Float64` — `double`.

We recommend that you store data in integer form whenever possible. For example, convert fixed precision numbers to integer values, such as monetary amounts or page load times in milliseconds.

Aliases:

- `Float32` — `FLOAT`.
- `Float64` — `DOUBLE`.

When creating tables, numeric parameters for floating point numbers can be set (e.g. `FLOAT(12)`, `FLOAT(15, 22)`, `DOUBLE(12)`, `DOUBLE(4, 18)`), but ClickHouse ignores them.

## Using Floating-point Numbers

- Computations with floating-point numbers might produce a rounding error.

```
SELECT 1 - 0.9
```

```
minus(1, 0.9)|
0.0999999999999998 |
```

- The result of the calculation depends on the calculation method (the processor type and architecture of the computer system).
- Floating-point calculations might result in numbers such as infinity (`Inf`) and “not-a-number” (`NaN`). This should be taken into account when processing the results of calculations.
- When parsing floating-point numbers from text, the result might not be the nearest machine-representable number.

## `Nan` and `Inf`

In contrast to standard SQL, ClickHouse supports the following categories of floating-point numbers:

- `Inf` — Infinity.

```
SELECT 0.5 / 0
```

```
divide(0.5, 0)|
inf |
```

- `-Inf` — Negative infinity.

```
SELECT -0.5 / 0
```

```
divide(-0.5, 0)|
-inf |
```

- `Nan` — Not a number.

```
SELECT 0 / 0
```

```
divide(0, 0)|
nan |
```

See the rules for `Nan` sorting in the section [ORDER BY clause](#).

# Decimal(P, S), Decimal32(S), Decimal64(S), Decimal128(S), Decimal256(S)

Signed fixed-point numbers that keep precision during add, subtract and multiply operations. For division least significant digits are discarded (not rounded).

## Parameters

- P - precision. Valid range: [ 1 : 76 ]. Determines how many decimal digits number can have (including fraction).
- S - scale. Valid range: [ 0 : P ]. Determines how many decimal digits fraction can have.

Depending on P parameter value Decimal(P, S) is a synonym for:

- P from [ 1 : 9 ] - for Decimal32(S)
- P from [ 10 : 18 ] - for Decimal64(S)
- P from [ 19 : 38 ] - for Decimal128(S)
- P from [ 39 : 76 ] - for Decimal256(S)

## Decimal Value Ranges

- Decimal32(S) - (  $-1 * 10^{(9 - S)}$ ,  $1 * 10^{(9 - S)}$  )
- Decimal64(S) - (  $-1 * 10^{(18 - S)}$ ,  $1 * 10^{(18 - S)}$  )
- Decimal128(S) - (  $-1 * 10^{(38 - S)}$ ,  $1 * 10^{(38 - S)}$  )
- Decimal256(S) - (  $-1 * 10^{(76 - S)}$ ,  $1 * 10^{(76 - S)}$  )

For example, Decimal32(4) can contain numbers from -99999.9999 to 99999.9999 with 0.0001 step.

## Internal Representation

Internally data is represented as normal signed integers with respective bit width. Real value ranges that can be stored in memory are a bit larger than specified above, which are checked only on conversion from a string.

Because modern CPUs do not support 128-bit integers natively, operations on Decimal128 are emulated. Because of this Decimal128 works significantly slower than Decimal32/Decimal64.

## Operations and Result Type

Binary operations on Decimal result in wider result type (with any order of arguments).

- `Decimal64(S1) <op> Decimal32(S2) -> Decimal64(S)`
- `Decimal128(S1) <op> Decimal32(S2) -> Decimal128(S)`
- `Decimal128(S1) <op> Decimal64(S2) -> Decimal128(S)`
- `Decimal256(S1) <op> Decimal<32|64|128>(S2) -> Decimal256(S)`

Rules for scale:

- add, subtract:  $S = \max(S1, S2)$ .
- multiply:  $S = S1 + S2$ .
- divide:  $S = S1$ .

For similar operations between Decimal and integers, the result is Decimal of the same size as an argument.

Operations between Decimal and Float32/Float64 are not defined. If you need them, you can explicitly cast one of argument using toDecimal32, toDecimal64, toDecimal128 or toFloat32, toFloat64 builtins. Keep in mind that the result will lose precision and type conversion is a computationally expensive operation.

Some functions on Decimal return result as Float64 (for example, var or stddev). Intermediate calculations might still be performed in Decimal, which might lead to different results between Float64 and Decimal inputs with the same values.

## Overflow Checks

During calculations on Decimal, integer overflows might happen. Excessive digits in a fraction are discarded (not rounded). Excessive digits in integer part will lead to an exception.

```
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.

Overflow checks lead to operations slowdown. If it is known that overflows are not possible, it makes sense to disable checks using `decimal_check_overflow` setting. When checks are disabled and overflow happens, the result will be incorrect:

```
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

Overflow checks happen not only on arithmetic operations but also on value comparison:

```
SELECT toDecimal32(1, 8) < 100
```

DB::Exception: Can't compare.

### See also

- [isDecimalOverflow](#)
- [countDigits](#)

# Boolean Values

There is no separate type for boolean values. Use UInt8 type, restricted to the values 0 or 1.

# String

Strings of an arbitrary length. The length is not limited. The value can contain an arbitrary set of bytes, including null bytes.

The String type replaces the types VARCHAR, BLOB, CLOB, and others from other DBMSs.

When creating tables, numeric parameters for string fields can be set (e.g. VARCHAR(255)), but ClickHouse ignores them.

# Encodings

ClickHouse does not have the concept of encodings. Strings can contain an arbitrary set of bytes, which are stored and output as-is.

If you need to store texts, we recommend using UTF-8 encoding. At the very least, if your terminal uses UTF-8 (as recommended), you can read and write your values without making conversions.

Similarly, certain functions for working with strings have separate variations that work under the assumption that the string contains a set of bytes representing a UTF-8 encoded text.

For example, the `length` function calculates the string length in bytes, while the `lengthUTF8` function calculates the string length in Unicode code points, assuming that the value is UTF-8 encoded.

# Fixedstring

A fixed-length string of `N` bytes (neither characters nor code points).

To declare a column of FixedString type, use the following syntax:

```
<column_name> FixedString(N)
```

Where `N` is a natural number.

The `FixedString` type is efficient when data has the length of precisely `N` bytes. In all other cases, it is likely to reduce efficiency.

Examples of the values that can be efficiently stored in `FixedString`-typed columns:

- The binary representation of IP addresses (`FixedString(16)` for IPv6).
- Language codes (`ru_RU`, `en_US` ... ).
- Currency codes (`USD`, `RUB` ... ).
- Binary representation of hashes (`FixedString(16)` for MD5, `FixedString(32)` for SHA256).

To store UUID values, use the `UUID` data type.

When inserting the data, ClickHouse:

- Completes a string with null bytes if the string contains fewer than `N` bytes.
- Throws the `Too large value for FixedString(N)` exception if the string contains more than `N` bytes.

When selecting the data, ClickHouse does not remove the null bytes at the end of the string. If you use the `WHERE` clause, you should add null bytes manually to match the `FixedString` value. The following example illustrates how to use the `WHERE` clause with `FixedString`.

Let's consider the following table with the single `FixedString(2)` column:

name
b

The query `SELECT * FROM FixedStringTable WHERE a = 'b'` does not return any data as a result. We should complement the filter pattern with null bytes.

```
SELECT * FROM FixedStringTable  
WHERE a = 'b\0'
```

a
b

This behaviour differs from MySQL for the `CHAR` type (where strings are padded with spaces, and the spaces are removed for output).

Note that the length of the `FixedString(N)` value is constant. The `length` function returns `N` even if the `FixedString(N)` value is filled only with null bytes, but the `empty` function returns `1` in this case.

## UUID

A universally unique identifier (UUID) is a 16-byte number used to identify records. For detailed information about the UUID, see [Wikipedia](#).

The example of UUID type value is represented below:

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

If you do not specify the UUID column value when inserting a new record, the UUID value is filled with zero:

```
00000000-0000-0000-0000-000000000000
```

## How to Generate

To generate the UUID value, ClickHouse provides the `generateUUIDv4` function.

## Usage Example

### Example 1

This example demonstrates creating a table with the UUID type column and inserting a value into the table.

```
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

## Example 2

In this example, the UUID column value is not specified when inserting a new record.

```
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

## Restrictions

The UUID data type only supports functions which [String](#) data type also supports (for example, [min](#), [max](#), and [count](#)).

The UUID data type is not supported by arithmetic operations (for example, [abs](#)) or aggregate functions, such as [sum](#) and [avg](#).

## Date

A date. Stored in two bytes as the number of days since 1970-01-01 (unsigned). Allows storing values from just after the beginning of the Unix Epoch to the upper threshold defined by a constant at the compilation stage (currently, this is until the year 2149, but the final fully-supported year is 2148).

The date value is stored without the time zone.

## Example

Creating a table with a Date-type column and inserting data into it:

```
CREATE TABLE dt
(
    `timestamp` Date,
    `event_id` UInt8
)
ENGINE = TinyLog;
```

```
INSERT INTO dt VALUES (1546300800, 1), ('2019-01-01', 2);
SELECT * FROM dt;
```

timestamp	event_id
2019-01-01	1
2019-01-01	2

## See Also

- [Functions for working with dates and times](#)
- [Operators for working with dates and times](#)
- [DateTime data type](#)

## Date32

A date. Supports the date range same with [Datetime64](#). Stored in four bytes as the number of days since 1925-01-01. Allows storing values till 2283-11-11.

### Examples

Creating a table with a Date32-type column and inserting data into it:

```
CREATE TABLE new
(
    `timestamp` Date32,
    `event_id` UInt8
)
ENGINE = TinyLog;
```

```
INSERT INTO new VALUES (4102444800, 1), ('2100-01-01', 2);
SELECT * FROM new;
```

timestamp	event_id
2100-01-01	1
2100-01-01	2

## See Also

- [toDate32](#)
- [toDate32OrZero](#)
- [toDate32OrNull](#)

## Datetime

Allows to store an instant in time, that can be expressed as a calendar date and a time of a day.

Syntax:

```
DateTime([timezone])
```

Supported range of values: [1970-01-01 00:00:00, 2105-12-31 23:59:59].

Resolution: 1 second.

# Usage Remarks

The point in time is saved as a [Unix timestamp](#), regardless of the time zone or daylight saving time. The time zone affects how the values of the `DateTime` type values are displayed in text format and how the values specified as strings are parsed ('2020-01-01 05:00:01').

Timezone agnostic unix timestamp is stored in tables, and the timezone is used to transform it to text format or back during data import/export or to make calendar calculations on the values (example: `toDate`, `toHour` functions et cetera). The time zone is not stored in the rows of the table (or in resultset), but is stored in the column metadata.

A list of supported time zones can be found in the [IANA Time Zone Database](#) and also can be queried by `SELECT * FROM system.time_zones`. [The list](#) is also available at Wikipedia.

You can explicitly set a time zone for `DateTime`-type columns when creating a table. Example: `DateTime('UTC')`. If the time zone isn't set, ClickHouse uses the value of the `timezone` parameter in the server settings or the operating system settings at the moment of the ClickHouse server start.

The [clickhouse-client](#) applies the server time zone by default if a time zone isn't explicitly set when initializing the data type. To use the client time zone, run `clickhouse-client` with the `--use_client_time_zone` parameter.

ClickHouse outputs values depending on the value of the `date_time_output_format` setting. `YYYY-MM-DD hh:mm:ss` text format by default. Additionally you can change the output with the `formatDateTime` function.

When inserting data into ClickHouse, you can use different formats of date and time strings, depending on the value of the `date_time_input_format` setting.

## Examples

### 1. Creating a table with a `DateTime`-type column and inserting data into it:

```
CREATE TABLE dt
(
    `timestamp` DateTime('Europe/Moscow'),
    `event_id` UInt8
)
ENGINE = TinyLog;
```

```
INSERT INTO dt Values (1546300800, 1), ('2019-01-01 00:00:00', 2);
```

```
SELECT * FROM dt;
```

timestamp	event_id
2019-01-01 03:00:00	1
2019-01-01 00:00:00	2

- When inserting datetime as an integer, it is treated as Unix Timestamp (UTC). `1546300800` represents `'2019-01-01 00:00:00'` UTC. However, as `timestamp` column has `Europe/Moscow` (UTC+3) timezone specified, when outputting as string the value will be shown as `'2019-01-01 03:00:00'`
- When inserting string value as datetime, it is treated as being in column timezone. `'2019-01-01 00:00:00'` will be treated as being in `Europe/Moscow` timezone and saved as `1546290000`.

### 2. Filtering on `DateTime` values

```
SELECT * FROM dt WHERE timestamp = toDateTime('2019-01-01 00:00:00', 'Europe/Moscow')
```

timestamp	event_id
2019-01-01 00:00:00	2

DateTime column values can be filtered using a string value in WHERE predicate. It will be converted to DateTime automatically:

```
SELECT * FROM dt WHERE timestamp = '2019-01-01 00:00:00'
```

timestamp	event_id
2019-01-01 03:00:00	1

### 3. Getting a time zone for a DateTime-type column:

```
SELECT toDateTime(now(), 'Europe/Moscow') AS column, toTypeName(column) AS x
```

column	x
2019-10-16 04:12:04	DateTime('Europe/Moscow')

### 4. Timezone conversion

```
SELECT
toDateTime(timestamp, 'Europe/London') as lon_time,
toDateTime(timestamp, 'Europe/Moscow') as mos_time
FROM dt
```

lon_time	mos_time
2019-01-01 00:00:00	2019-01-01 03:00:00
2018-12-31 21:00:00	2019-01-01 00:00:00

As timezone conversion only changes the metadata, the operation has no computation cost.

## Limitations on timezones support

Some timezones may not be supported completely. There are a few cases:

If the offset from UTC is not a multiple of 15 minutes, the calculation of hours and minutes can be incorrect. For example, the time zone in Monrovia, Liberia has offset UTC -0:44:30 before 7 Jan 1972. If you are doing calculations on the historical time in Monrovia timezone, the time processing functions may give incorrect results. The results after 7 Jan 1972 will be correct nevertheless.

If the time transition (due to daylight saving time or for other reasons) was performed at a point of time that is not a multiple of 15 minutes, you can also get incorrect results at this specific day.

Non-monotonic calendar dates. For example, in Happy Valley - Goose Bay, the time was transitioned one hour backwards at 00:01:00 7 Nov 2010 (one minute after midnight). So after 6th Nov has ended, people observed a whole one minute of 7th Nov, then time was changed back to 23:01 6th Nov and after another 59 minutes the 7th Nov started again. ClickHouse does not (yet) support this kind of fun. During these days the results of time processing functions may be slightly incorrect.

Similar issue exists for Casey Antarctic station in year 2010. They changed time three hours back at 5 Mar, 02:00. If you are working in antarctic station, please don't afraid to use ClickHouse. Just make sure you set timezone to UTC or be aware of inaccuracies.

Time shifts for multiple days. Some pacific islands changed their timezone offset from UTC+14 to UTC-12. That's alright but some inaccuracies may present if you do calculations with their timezone for historical time points at the days of conversion.

## See Also

- [Type conversion functions](#)
- [Functions for working with dates and times](#)
- [Functions for working with arrays](#)
- [The `date\_time\_input\_format` setting](#)
- [The `date\_time\_output\_format` setting](#)
- [The `timezone` server configuration parameter](#)
- [Operators for working with dates and times](#)
- [The `Date` data type](#)

## Datetime64

Allows to store an instant in time, that can be expressed as a calendar date and a time of a day, with defined sub-second precision

Tick size (precision):  $10^{-\text{precision}}$  seconds. Valid range: [ 0 : 9 ].

Typically are used - 3 (milliseconds), 6 (microseconds), 9 (nanoseconds).

### Syntax:

```
DateTime64(precision, [timezone])
```

Internally, stores data as a number of ‘ticks’ since epoch start (1970-01-01 00:00:00 UTC) as Int64. The tick resolution is determined by the precision parameter. Additionally, the `DateTime64` type can store time zone that is the same for the entire column, that affects how the values of the `DateTime64` type values are displayed in text format and how the values specified as strings are parsed ('2020-01-01 05:00:01.000'). The time zone is not stored in the rows of the table (or in resultset), but is stored in the column metadata. See details in [DateTime](#).

Supported range from January 1, 1925 till November 11, 2283.

## Examples

1. Creating a table with `DateTime64`-type column and inserting data into it:

```
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

- When inserting datetime as an integer, it is treated as an appropriately scaled Unix Timestamp (UTC). `1546300800000` (with precision 3) represents `'2019-01-01 00:00:00'` UTC. However, as `timestamp` column has Europe/Moscow (UTC+3) timezone specified, when outputting as a string the value will be shown as `'2019-01-01 03:00:00'`.
- When inserting string value as datetime, it is treated as being in column timezone. `'2019-01-01 00:00:00'` will be treated as being in Europe/Moscow timezone and stored as `1546290000000`.

## 2. Filtering on DateTime64 values

```
SELECT * FROM dt WHERE timestamp = toDateTime64('2019-01-01 00:00:00', 3, 'Europe/Moscow');
```

timestamp	event_id
2019-01-01 00:00:00.000	2

Unlike `DateTime`, `DateTime64` values are not converted from `String` automatically.

## 3. Getting a time zone for a DateTime64-type value:

```
SELECT toDateTime64(now(), 3, 'Europe/Moscow') AS column, toTypeName(column) AS x;
```

column	x
2019-10-16 04:12:04.000	DateTime64(3, 'Europe/Moscow')

## 4. Timezone conversion

```
SELECT
toDateTime64(timestamp, 3, 'Europe/London') as lon_time,
toDateTime64(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

## See Also

- Type conversion functions
- Functions for working with dates and times
- Functions for working with arrays

- The `date_time_input_format` setting
- The `date_time_output_format` setting
- The `timezone` server configuration parameter
- Operators for working with dates and times
- Date data type
- DateTime data type

## Enum

Enumerated type consisting of named values.

Named values must be declared as 'string' = integer pairs. ClickHouse stores only numbers, but supports operations with the values through their names.

ClickHouse supports:

- 8-bit `Enum`. It can contain up to 256 values enumerated in the [-128, 127] range.
- 16-bit `Enum`. It can contain up to 65536 values enumerated in the [-32768, 32767] range.

ClickHouse automatically chooses the type of `Enum` when data is inserted. You can also use `Enum8` or `Enum16` types to be sure in the size of storage.

## Usage Examples

Here we create a table with an `Enum8('hello' = 1, 'world' = 2)` type column:

```
CREATE TABLE t_enum
(
    x Enum('hello' = 1, 'world' = 2)
)
ENGINE = TinyLog
```

Column `x` can only store values that are listed in the type definition: `'hello'` or `'world'`. If you try to save any other value, ClickHouse will raise an exception. 8-bit size for this `Enum` is chosen automatically.

```
INSERT INTO t_enum VALUES ('hello'), ('world'), ('hello')
```

```
Ok.
```

```
INSERT INTO t_enum values('a')
```

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

When you query data from the table, ClickHouse outputs the string values from `Enum`.

```
SELECT * FROM t_enum
```

x
hello
world
hello

If you need to see the numeric equivalents of the rows, you must cast the `Enum` value to integer type.

```
SELECT CAST(x, 'Int8') FROM t_enum
```

CAST(x, 'Int8')
1
2
1

To create an `Enum` value in a query, you also need to use `CAST`.

```
SELECT toTypeName(CAST('a', 'Enum(\\'a\\\') = 1, \\'b\\\') = 2'))
```

```
toTypeName(CAST('a', 'Enum(\\'a\\\') = 1, \\'b\\\') = 2'))
```

```
Enum8('a' = 1, 'b' = 2)
```

## General Rules and Usage

Each of the values is assigned a number in the range -128 ... 127 for `Enum8` or in the range -32768 ... 32767 for `Enum16`. All the strings and numbers must be different. An empty string is allowed. If this type is specified (in a table definition), numbers can be in an arbitrary order. However, the order does not matter.

Neither the string nor the numeric value in an `Enum` can be `NULL`.

An `Enum` can be contained in `Nullable` type. So if you create a table using the query

```
CREATE TABLE t_enum_nullable
(
    x Nullable( Enum8('hello' = 1, 'world' = 2) )
)
ENGINE = TinyLog
```

it can store not only 'hello' and 'world', but `NULL`, as well.

```
INSERT INTO t_enum_nullable Values('hello'),('world'),(NULL)
```

In RAM, an `Enum` column is stored in the same way as `Int8` or `Int16` of the corresponding numerical values.

When reading in text form, ClickHouse parses the value as a string and searches for the corresponding string from the set of `Enum` values. If it is not found, an exception is thrown. When reading in text format, the string is read and the corresponding numeric value is looked up. An exception will be thrown if it is not found.

When writing in text form, it writes the value as the corresponding string. If column data contains garbage (numbers that are not from the valid set), an exception is thrown. When reading and writing in binary form, it works the same way as for `Int8` and `Int16` data types.

The implicit default value is the value with the lowest number.

During `ORDER BY`, `GROUP BY`, `IN`, `DISTINCT` and so on, Enums behave the same way as the corresponding numbers. For example, `ORDER BY` sorts them numerically. Equality and comparison operators work the same way on Enums as they do on the underlying numeric values.

Enum values cannot be compared with numbers. Enums can be compared to a constant string. If the string compared to is not a valid value for the Enum, an exception will be thrown. The `IN` operator is supported with the Enum on the left-hand side and a set of strings on the right-hand side. The strings are the values of the corresponding Enum.

Most numeric and string operations are not defined for Enum values, e.g. adding a number to an Enum or concatenating a string to an Enum.

However, the Enum has a natural `toString` function that returns its string value.

Enum values are also convertible to numeric types using the `toT` function, where T is a numeric type. When T corresponds to the enum's underlying numeric type, this conversion is zero-cost.

The Enum type can be changed without cost using `ALTER`, if only the set of values is changed. It is possible to both add and remove members of the Enum using `ALTER` (removing is safe only if the removed value has never been used in the table). As a safeguard, changing the numeric value of a previously defined Enum member will throw an exception.

Using `ALTER`, it is possible to change an `Enum8` to an `Enum16` or vice versa, just like changing an `Int8` to `Int16`.

## LowCardinality Data Type

Changes the internal representation of other data types to be dictionary-encoded.

### Syntax

```
LowCardinality(data_type)
```

#### Parameters

- `data_type` — `String`, `FixedString`, `Date`, `DateTime`, and numbers excepting `Decimal`. `LowCardinality` is not efficient for some data types, see the [allow\\_suspicious\\_low\\_cardinality\\_types](#) setting description.

### Description

`LowCardinality` is a superstructure that changes a data storage method and rules of data processing.

ClickHouse applies [dictionary coding](#) to `LowCardinality`-columns. Operating with dictionary encoded data significantly increases performance of `SELECT` queries for many applications.

The efficiency of using `LowCardinality` data type depends on data diversity. If a dictionary contains less than 10,000 distinct values, then ClickHouse mostly shows higher efficiency of data reading and storing. If a dictionary contains more than 100,000 distinct values, then ClickHouse can perform worse in comparison with using ordinary data types.

Consider using `LowCardinality` instead of `Enum` when working with strings. `LowCardinality` provides more flexibility in use and often reveals the same or higher efficiency.

### Example

Create a table with a `LowCardinality`-column:

```
CREATE TABLE lc_t
(
    `id` UInt16,
    `strings` LowCardinality(String)
)
ENGINE = MergeTree()
ORDER BY id
```

## Related Settings and Functions

Settings:

- [low\\_cardinality\\_max\\_dictionary\\_size](#)
- [low\\_cardinality\\_use\\_single\\_dictionary\\_for\\_part](#)
- [low\\_cardinality\\_allow\\_in\\_native\\_format](#)
- [allow\\_suspicious\\_low\\_cardinality\\_types](#)
- [output\\_format\\_arrow\\_low\\_cardinality\\_as\\_dictionary](#)

Functions:

- [toLowCardinality](#)

## See Also

- [Reducing ClickHouse Storage Cost with the Low Cardinality Type – Lessons from an Instana Engineer.](#)
- [String Optimization \(video presentation in Russian\). Slides in English.](#)

## Array(t)

An array of `T`-type items. `T` can be any data type, including an array.

### Creating an Array

You can use a function to create an array:

```
array(T)
```

You can also use square brackets.

```
[]
```

Example of creating an array:

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

```
x-----toTypeName(array(1, 2))-----
[1,2] | Array(UInt8) |
```

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

```
x-----toTypeName([1, 2])-----  
| [1,2] | Array(UInt8) |
```

## Working with Data Types

The maximum size of an array is limited to one million elements.

When creating an array on the fly, ClickHouse automatically defines the argument type as the narrowest data type that can store all the listed arguments. If there are any **Nullable** or literal **NULL** values, the type of an array element also becomes **Nullable**.

If ClickHouse couldn't determine the data type, it generates an exception. For instance, this happens when trying to create an array with strings and numbers simultaneously (`SELECT array(1, 'a')`).

Examples of automatic data type detection:

```
SELECT array(1, 2, NULL) AS x, toTypeName(x)
```

```
x-----toTypeName(array(1, 2, NULL))-----  
| [1,2,NULL] | Array(Nullable(UInt8)) |
```

If you try to create an array of incompatible data types, ClickHouse throws an exception:

```
SELECT array(1, 'a')
```

```
Received exception from server (version 1.1.54388):  
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.
```

## Array Size

It is possible to find the size of an array by using the `size0` subcolumn without reading the whole column. For multi-dimensional arrays you can use `sizeN-1`, where N is the wanted dimension.

### Example

Query:

```
CREATE TABLE t_arr (`arr` Array(Array(Array(UInt32)))) ENGINE = MergeTree ORDER BY tuple();  
  
INSERT INTO t_arr VALUES ([[12, 13, 0, 1],[12]]);  
  
SELECT arr.size0, arr.size1, arr.size2 FROM t_arr;
```

Result:

```
arr.size0-----arr.size1-----arr.size2-----  
| 1 | [2] | | [[4,1]] | |
```

## AggregateFunction

Aggregate functions can have an implementation-defined intermediate state that can be serialized to an `AggregateFunction(...)` data type and stored in a table, usually, by means of [a materialized view](#). The common way to produce an aggregate function state is by calling the aggregate function with the `-State` suffix. To get the final result of aggregation in the future, you must use the same aggregate function with the `-Mergesuffix`.

`AggregateFunction(name, types_of_arguments...)` — parametric data type.

## Parameters

- Name of the aggregate function.

If the function is parametric, specify its parameters too.

- Types of the aggregate function arguments.

## Example

```
CREATE TABLE t
(
    column1 AggregateFunction(uniq, UInt64),
    column2 AggregateFunction(anyIf, String, UInt8),
    column3 AggregateFunction(quantiles(0.5, 0.9), UInt64)
) ENGINE = ...
```

`uniq`, `anyIf` (`any+If`) and `quantiles` are the aggregate functions supported in ClickHouse.

## Usage

### Data Insertion

To insert data, use `INSERT SELECT` with aggregate `-State-` functions.

### Function examples

```
uniqState(UserID)
quantilesState(0.5, 0.9)(SendTiming)
```

In contrast to the corresponding functions `uniq` and `quantiles`, `-State-` functions return the state, instead of the final value. In other words, they return a value of `AggregateFunction` type.

In the results of `SELECT` query, the values of `AggregateFunction` type have implementation-specific binary representation for all of the ClickHouse output formats. If dump data into, for example, `TabSeparated` format with `SELECT` query, then this dump can be loaded back using `INSERT` query.

### Data Selection

When selecting data from `AggregatingMergeTree` table, use `GROUP BY` clause and the same aggregate functions as when inserting data, but using `-Mergesuffix`.

An aggregate function with `-Merge` suffix takes a set of states, combines them, and returns the result of complete data aggregation.

For example, the following two queries return the same result:

```
SELECT uniq(UserID) FROM table
```

```
SELECT uniqMerge(state) FROM (SELECT uniqState(UserID) AS state FROM table GROUP BY RegionID)
```

# Usage Example

See [AggregatingMergeTree](#) engine description.

## Nested Data Structures

### Nested

#### Nested(name1 Type1, Name2 Type2, ...)

A nested data structure is like a table inside a cell. The parameters of a nested data structure – the column names and types – are specified the same way as in a [CREATE TABLE](#) query. Each table row can correspond to any number of rows in a nested data structure.

Example:

```
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)
```

This example declares the `Goals` nested data structure, which contains data about conversions (goals reached). Each row in the ‘visits’ table can correspond to zero or any number of conversions.

When `flatten_nested` is set to 0 (which is not by default), arbitrary levels of nesting are supported.

In most cases, when working with a nested data structure, its columns are specified with column names separated by a dot. These columns make up an array of matching types. All the column arrays of a single nested data structure have the same length.

Example:

```
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 22:18:51']
[]	[]
[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']

It is easiest to think of a nested data structure as a set of multiple column arrays of the same length.

The only place where a SELECT query can specify the name of an entire nested data structure instead of individual columns is the ARRAY JOIN clause. For more information, see “ARRAY JOIN clause”. Example:

```
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

You can't perform SELECT for an entire nested data structure. You can only explicitly list individual columns that are part of it.

For an INSERT query, you should pass all the component column arrays of a nested data structure separately (as if they were individual column arrays). During insertion, the system checks that they have the same length.

For a DESCRIBE query, the columns in a nested data structure are listed separately in the same way.

The ALTER query for elements in a nested data structure has limitations.

## Tuple(t1, T2, ...)

A tuple of elements, each having an individual [type](#).

Tuples are used for temporary column grouping. Columns can be grouped when an IN expression is used in a query, and for specifying certain formal parameters of lambda functions. For more information, see the sections [IN operators](#) and [Higher order functions](#).

Tuples can be the result of a query. In this case, for text formats other than JSON, values are comma-separated in brackets. In JSON formats, tuples are output as arrays (in square brackets).

## Creating a Tuple

You can use a function to create a tuple:

```
tuple(T1, T2, ...)
```

Example of creating a tuple:

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

```
x-----toTypeName(tuple(1, 'a'))-----  
(1,'a') | Tuple(UInt8, String) |
```

## Working with Data Types

When creating a tuple on the fly, ClickHouse automatically detects the type of each argument as the minimum of the types which can store the argument value. If the argument is [NULL](#), the type of the tuple element is [Nullable](#).

Example of automatic data type detection:

```
SELECT tuple(1, NULL) AS x, toTypeName(x)
```

```
x-----toTypeName(tuple(1, NULL))-----  
(1,NULL) | Tuple(UInt8, Nullable(Nothing)) |
```

## Addressing Tuple Elements

It is possible to read elements of named tuples using indexes and names:

```
CREATE TABLE named_tuples (`a` Tuple(s String, i Int64)) ENGINE = Memory;  
INSERT INTO named_tuples VALUES ('y', 10), ('x', -10);  
SELECT a.s FROM named_tuples;  
SELECT a.2 FROM named_tuples;
```

Result:

a.s
y
x

tupleElement(a, 2)
10
-10

## Nullable(**typename**)

Allows to store special marker (**NULL**) that denotes “missing value” alongside normal values allowed by **TypeName**. For example, a **Nullable(Int8)** type column can store **Int8** type values, and the rows that do not have a value will store **NULL**.

For a **TypeName**, you can't use composite data types **Array** and **Tuple**. Composite data types can contain **Nullable** type values, such as **Array(Nullable(Int8))**.

A **Nullable** type field can't be included in table indexes.

**NULL** is the default value for any **Nullable** type, unless specified otherwise in the ClickHouse server configuration.

## Storage Features

To store **Nullable** type values in a table column, ClickHouse uses a separate file with **NULL** masks in addition to normal file with values. Entries in masks file allow ClickHouse to distinguish between **NULL** and a default value of corresponding data type for each table row. Because of an additional file, **Nullable** column consumes additional storage space compared to a similar normal one.

### Note

Using **Nullable** almost always negatively affects performance, keep this in mind when designing your databases.

## Finding **NULL**

It is possible to find **NULL** values in a column by using **null** subcolumn without reading the whole column. It returns **1** if the corresponding value is **NULL** and **0** otherwise.

### Example

Query:

```
CREATE TABLE nullable(`n` Nullable(UInt32)) ENGINE = MergeTree ORDER BY tuple();
INSERT INTO nullable VALUES (1) (NULL) (2) (NULL);
SELECT n.null FROM nullable;
```

Result:

n.null
0
1
0
1

## Usage Example

```
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
```

plus(x, y)
NULL
5

## Special Data Types

Special data type values can't be serialized for saving in a table or output in query results, but can be used as an intermediate result during query execution.

## Expression

Expressions are used for representing lambdas in high-order functions.

## Set

Used for the right half of an **IN** expression.

## Nothing

The only purpose of this data type is to represent cases where a value is not expected. So you can't create a Nothing type value.

For example, literal **NULL** has type of `Nullable(Nothing)`. See more about [Nullable](#).

The Nothing type can also be used to denote empty arrays:

```
SELECT toTypeName(array())
```

toTypeName(array())
Array(Nothing)

# Interval

The family of data types representing time and date intervals. The resulting types of the **INTERVAL** operator.

## Warning

Interval data type values can't be stored in tables.

Structure:

- Time interval as an unsigned integer value.
- Type of an interval.

Supported interval types:

- **SECOND**
- **MINUTE**
- **HOUR**
- **DAY**
- **WEEK**
- **MONTH**
- **QUARTER**
- **YEAR**

For each interval type, there is a separate data type. For example, the **DAY** interval corresponds to the **IntervalDay** data type:

```
SELECT toTypeName(INTERVAL 4 DAY)
```

```
toTypeName(toIntervalDay(4))  
IntervalDay
```

## Usage Remarks

You can use **Interval**-type values in arithmetical operations with **Date** and **DateTime**-type values. For example, you can add 4 days to the current time:

```
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 |
```

Intervals with different types can't be combined. You can't use intervals like `4 DAY 1 HOUR`. Specify intervals in units that are smaller or equal to the smallest unit of the interval, for example, the interval `1 day` and an `hour` interval can be expressed as `25 HOUR` or `90000 SECOND`.

You can't perform arithmetical operations with `Interval`-type values, but you can add intervals of different types consequently to values in `Date` or `DateTime` data types. For example:

```
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 |
```

The following query causes an exception:

```
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..

## See Also

- [INTERVAL operator](#)
- [toInterval type conversion functions](#)

## Domains

Domains are special-purpose types that add some extra features atop of existing base type, but leaving on-wire and on-disc format of the underlying data type intact. At the moment, ClickHouse does not support user-defined domains.

You can use domains anywhere corresponding base type can be used, for example:

- Create a column of a domain type
- Read/write values from/to domain column
- Use it as an index if a base type can be used as an index
- Call functions with values of domain column

## Extra Features of Domains

- Explicit column type name in `SHOW CREATE TABLE` or `DESCRIBE TABLE`
- Input from human-friendly format with `INSERT INTO domain_table(domain_column) VALUES(...)`
- Output to human-friendly format for `SELECT domain_column FROM domain_table`
- Loading data from an external source in the human-friendly format: `INSERT INTO domain_table FORMAT CSV`
- ...

## Limitations

- Can't convert index column of base type to domain type via `ALTER TABLE`.
- Can't implicitly convert string values into domain values when inserting data from another column or table.
- Domain adds no constraints on stored values.

## IPv4

`IPv4` is a domain based on `UInt32` type and serves as a typed replacement for storing IPv4 values. It provides compact storage with the human-friendly input-output format and column type information on inspection.

### Basic Usage

```
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				

OR you can use `IPv4` domain as a key:

```
CREATE TABLE hits (url String, from IPv4) ENGINE = MergeTree() ORDER BY from;
```

`IPv4` domain supports custom input format as `IPv4`-strings:

```
INSERT INTO hits (url, from) VALUES ('https://wikipedia.org', '116.253.40.133')('https://clickhouse.com', '183.247.232.58')('https://clickhouse.com/docs/en/', '116.106.34.242');
SELECT * FROM hits;
```

url	from
https://clickhouse.com/docs/en/	116.106.34.242
https://wikipedia.org	116.253.40.133
https://clickhouse.com	183.247.232.58

Values are stored in compact binary form:

```
SELECT toTypeName(from), hex(from) FROM hits LIMIT 1;
```

toTypeName(from)	hex(from)
IPv4	B7F7E83A

Domain values are not implicitly convertible to types other than `UInt32`.

If you want to convert `IPv4` value to a string, you have to do that explicitly with `IPv4NumToString()` function:

```
SELECT toTypeName(s), IPv4NumToString(from) as s FROM hits LIMIT 1;
```

```
toTypeName(IPv4NumToString(from))-----s-----  
String          | 183.247.232.58 |
```

Or cast to a UInt32 value:

```
SELECT toTypeName(i), CAST(from as UInt32) as i FROM hits LIMIT 1;
```

```
toTypeName(CAST(from, 'UInt32'))-----i-----  
UInt32          | 3086477370 |
```

## IPv6

IPv6 is a domain based on `FixedString(16)` type and serves as a typed replacement for storing IPv6 values. It provides compact storage with the human-friendly input-output format and column type information on inspection.

### Basic Usage

```
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				

OR you can use IPv6 domain as a key:

```
CREATE TABLE hits (url String, from IPv6) ENGINE = MergeTree() ORDER BY from;
```

IPv6 domain supports custom input as IPv6-strings:

```
INSERT INTO hits (url, from) VALUES ('https://wikipedia.org', '2a02:aa08:e000:3100::2')('https://clickhouse.com', '2001:44c8:129:2632:33:0:252:2')('https://clickhouse.com/docs/en/', '2a02:e980:1e::1');
```

```
SELECT * FROM hits;
```

url	from
https://clickhouse.com	2001:44c8:129:2632:33:0:252:2
https://clickhouse.com/docs/en/	2a02:e980:1e::1
https://wikipedia.org	2a02:aa08:e000:3100::2

Values are stored in compact binary form:

```
SELECT toTypeName(from), hex(from) FROM hits LIMIT 1;
```

```
toTypeName(from)---hex(from)
IPv6      | 200144C801292632003300002520002 |
```

Domain values are not implicitly convertible to types other than `FixedString(16)`.

If you want to convert IPv6 value to a string, you have to do that explicitly with `IPv6NumToString()` function:

```
SELECT toTypeName(s), IPv6NumToString(from) as s FROM hits LIMIT 1;
```

```
toTypeName(IPv6NumToString(from))---s
String          | 2001:44c8:129:2632:33:0:252:2 |
```

Or cast to a `FixedString(16)` value:

```
SELECT toTypeName(i), CAST(from as FixedString(16)) as i FROM hits LIMIT 1;
```

```
toTypeName(CAST(from, 'FixedString(16)'))---i
FixedString(16) | ♦♦♦ |
```

## Multiword Types

When creating tables, you can use data types with a name consisting of several words. This is implemented for better SQL compatibility.

### Multiword Types Support

Multiword types	Simple types
DOUBLE PRECISION	Float64
CHAR LARGE OBJECT	String
CHAR VARYING	String
CHARACTER LARGE OBJECT	String
CHARACTER VARYING	String
NCHAR LARGE OBJECT	String
NCHAR VARYING	String
NATIONAL CHARACTER LARGE OBJECT	String
NATIONAL CHARACTER VARYING	String
NATIONAL CHAR VARYING	String

Multiword types	Simple types
NATIONAL CHARACTER	String
NATIONAL CHAR	String
BINARY LARGE OBJECT	String
BINARY VARYING	String

## Geo Data Types

ClickHouse supports data types for representing geographical objects — locations, lands, etc.

### Warning

Currently geo data types are an experimental feature. To work with them you must set `allow_experimental_geo_types = 1`.

#### See Also

- [Representing simple geographical features](#).
- [allow\\_experimental\\_geo\\_types](#) setting.

## Point

Point is represented by its X and Y coordinates, stored as a [Tuple\(Float64, Float64\)](#).

#### Example

Query:

```
SET allow_experimental_geo_types = 1;
CREATE TABLE geo_point (p Point) ENGINE = Memory();
INSERT INTO geo_point VALUES((10, 10));
SELECT p, toTypeName(p) FROM geo_point;
```

Result:

p	toTypeName(p)
(10,10)	Point

## Ring

Ring is a simple polygon without holes stored as an array of points: [Array\(Point\)](#).

#### Example

Query:

```
SET allow_experimental_geo_types = 1;
CREATE TABLE geo_ring (r Ring) ENGINE = Memory();
INSERT INTO geo_ring VALUES([(0, 0), (10, 0), (10, 10), (0, 10)]);
SELECT r, toTypeName(r) FROM geo_ring;
```

Result:

```
r [(0,0),(10,0),(10,10),(0,10)] | Ring | toTypeName(r)
```

## Polygon

Polygon is a polygon with holes stored as an array of rings: [Array\(Ring\)](#). First element of outer array is the outer shape of polygon and all the following elements are holes.

### Example

This is a polygon with one hole:

```
SET allow_experimental_geo_types = 1;
CREATE TABLE geo_polygon (pg Polygon) ENGINE = Memory();
INSERT INTO geo_polygon VALUES([(20, 20), (50, 20), (50, 50), (20, 50)], [(30, 30), (50, 50), (50, 30)]);
SELECT pg, toTypeName(pg) FROM geo_polygon;
```

Result:

```
pg [(20,20),(50,20),(50,50),(20,50)],[(30,30),(50,50),(50,30)] | Polygon | toTypeName(pg)
```

## MultiPolygon

MultiPolygon consists of multiple polygons and is stored as an array of polygons: [Array\(Polygon\)](#).

### Example

This multipolygon consists of two separate polygons — the first one without holes, and the second with one hole:

```
SET allow_experimental_geo_types = 1;
CREATE TABLE geo_multipolygon (mpg MultiPolygon) ENGINE = Memory();
INSERT INTO geo_multipolygon VALUES([[(0, 0), (10, 0), (10, 10), (0, 10)]], [(20, 20), (50, 20), (50, 50), (20, 50)], [(30, 30), (50, 50), (50, 30)]);
SELECT mpg, toTypeName(mpg) FROM geo_multipolygon;
```

Result:

```
mpg
TypeName(mpg)
[[[(0,0),(10,0),(10,10),(0,10)]],[(20,20),(50,20),(50,50),(20,50)],[(30,30),(50,50),(50,30)]] | MultiPolygon
```

## Map(key, value)

`Map(key, value)` data type stores key:value pairs.

## Parameters

- `key` — The key part of the pair. `String`, `Integer`, `LowCardinality`, or `FixedString`.
- `value` — The value part of the pair. `String`, `Integer`, `Array`, `LowCardinality`, or `FixedString`.

To get the value from an `a Map('key', 'value')` column, use `a['key']` syntax. This lookup works now with a linear complexity.

## Examples

Consider the table:

```
CREATE TABLE table_map (a Map(String, UInt64)) ENGINE=Memory;
INSERT INTO table_map VALUES ('key1':1, 'key2':10), ('key1':2,'key2':20}, ('key1':3,'key2':30});
```

Select all `key2` values:

```
SELECT a['key2'] FROM table_map;
```

Result:

```
arrayElement(a, 'key2')—
  10 |
  20 |
  30 |
```

If there's no such `key` in the `Map()` column, the query returns zeros for numerical values, empty strings or empty arrays.

```
INSERT INTO table_map VALUES ('key3':100), ();
SELECT a['key3'] FROM table_map;
```

Result:

```
arrayElement(a, 'key3')—
  100 |
    0 |
```

```
arrayElement(a, 'key3')—
    0 |
    0 |
    0 |
```

## Convert Tuple to Map Type

You can cast `Tuple()` as `Map()` using `CAST` function:

```
SELECT CAST(([1, 2, 3], ['Ready', 'Steady', 'Go']), 'Map(UInt8, String)') AS map;
```

```
map—
{1:'Ready',2:'Steady',3:'Go'} |
```

# Map.keys and Map.values Subcolumns

To optimize `Map` column processing, in some cases you can use the `keys` and `values` subcolumns instead of reading the whole column.

## Example

Query:

```
CREATE TABLE t_map (`a` Map(String, UInt64)) ENGINE = Memory;  
INSERT INTO t_map VALUES (map('key1', 1, 'key2', 2, 'key3', 3));  
SELECT a.keys FROM t_map;  
SELECT a.values FROM t_map;
```

Result:

```
a.keys  
['key1','key2','key3'] |  
  
a.values  
[1,2,3] |
```

## See Also

- [map\(\)](#) function
- [CAST\(\)](#) function

# 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  $S_1 \text{ UNION ALL } S_2$  can be obtained by applying  $f$  to parts of the row set separately, and then again applying  $f$  to the results:  $f(S_1 \text{ UNION ALL } S_2) = f(f(S_1) \text{ UNION ALL } f(S_2))$ . This property guarantees that partial aggregation results are enough to compute the combined one, so we do not have to store and process any extra data.

The common way to produce an aggregate function value is by calling the aggregate function with the `-SimpleState` suffix.

The following aggregate functions are supported:

- `any`
- `anyLast`
- `min`
- `max`
- `sum`
- `sumWithOverflow`
- `groupBitAnd`

- `groupBitOr`
- `groupBitXor`
- `groupArrayArray`
- `groupUniqArrayArray`
- `sumMap`
- `minMap`
- `maxMap`

## Note

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 simple (id UInt64, val SimpleAggregateFunction(sum, Double)) ENGINE=AggregatingMergeTree ORDER BY id;
```

# Operators

ClickHouse transforms operators to their corresponding functions at the query parsing stage according to their priority, precedence, and associativity.

## Access Operators

`a[N]` – Access to an element of an array. The `arrayElement(a, N)` function.

`a.N` – Access to a tuple element. The `tupleElement(a, N)` function.

## Numeric Negation Operator

`-a` – The `negate(a)` function.

For tuple negation: `tupleNegate`.

## Multiplication and Division Operators

`a * b` – The `multiply(a, b)` function.

For multiplying tuple by number: `tupleMultiplyByNumber`, for scalar product: `dotProduct`.

`a / b` – The `divide(a, b)` function.

For dividing tuple by number: `tupleDivideByNumber`.

`a % b` – The `modulo(a, b)` function.

## Addition and Subtraction Operators

`a + b` – The `plus(a, b)` function.

For tuple addition: `tuplePlus`.

`a - b` – The `minus(a, b)` function.

For tuple subtraction: `tupleMinus`.

## Comparison Operators

`a = b` – The `equals(a, b)` function.

`a == b` – The `equals(a, b)` function.

`a != b` – The `notEquals(a, b)` function.

`a <> b` – The `notEquals(a, b)` function.

`a <= b` – The `lessOrEquals(a, b)` function.

`a >= b` – The `greaterOrEquals(a, b)` function.

`a < b` – The `less(a, b)` function.

`a > b` – The `greater(a, b)` function.

`a LIKE s` – The `like(a, b)` function.

`a NOT LIKE s` – The `notLike(a, b)` function.

`a ILIKE s` – The `ilike(a, b)` function.

`a BETWEEN b AND c` – The same as `a >= b AND a <= c`.

`a NOT BETWEEN b AND c` – The same as `a < b OR a > c`.

## Operators for Working with Data Sets

See [IN operators](#).

`a IN ...` – The `in(a, b)` function.

`a NOT IN ...` – The `notIn(a, b)` function.

`a GLOBAL IN ...` – The `globalIn(a, b)` function.

`a GLOBAL NOT IN ...` – The `globalNotIn(a, b)` function.

`a = ANY (subquery)` – The `in(a, subquery)` function.

`a != ANY (subquery)` – The same as `a NOT IN (SELECT singleValueOrNull(*) FROM subquery)`.

`a = ALL (subquery)` – The same as `a IN (SELECT singleValueOrNull(*) FROM subquery)`.

`a != ALL (subquery)` – The `notIn(a, subquery)` function.

## Examples

Query with ALL:

```
SELECT number AS a FROM numbers(10) WHERE a > ALL (SELECT number FROM numbers(3, 3));
```

Result:

a
6
7
8
9

Query with ANY:

```
SELECT number AS a FROM numbers(10) WHERE a > ANY (SELECT number FROM numbers(3, 3));
```

Result:

a
4
5
6
7
8
9

## Operators for Working with Dates and Times

### EXTRACT

```
EXTRACT(part FROM date);
```

Extract parts from a given date. For example, you can retrieve a month from a given date, or a second from a time.

The `part` parameter specifies which part of the date to retrieve. The following values are available:

- `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.

The `part` parameter is case-insensitive.

The `date` parameter specifies the date or the time to process. Either `Date` or `DateTime` type is supported.

Examples:

```
SELECT EXTRACT(DAY FROM toDate('2017-06-15'));
SELECT EXTRACT(MONTH FROM toDate('2017-06-15'));
SELECT EXTRACT(YEAR FROM toDate('2017-06-15'));
```

In the following example we create a table and insert into it a value with the `DateTime` type.

```
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

You can see more examples in [tests](#).

## INTERVAL

Creates an `Interval`-type value that should be used in arithmetical operations with `Date` and `DateTime`-type values.

Types of intervals:

- SECOND
- MINUTE
- HOUR
- DAY
- WEEK
- MONTH
- QUARTER
- YEAR

You can also use a string literal when setting the `INTERVAL` value. For example, `INTERVAL 1 HOUR` is identical to the `INTERVAL '1 hour'` or `INTERVAL '1' hour`.

## Warning

Intervals with different types can't be combined. You can't use expressions like `INTERVAL 4 DAY 1 HOUR`. Specify intervals in units that are smaller or equal to the smallest unit of the interval, for example, `INTERVAL 25 HOUR`. You can use consecutive operations, like in the example below.

Examples:

```
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))—  
2020-11-03 22:09:50 | 2020-11-08 01:09:50 |
```

```
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))—  
2020-11-03 22:12:10 | 2020-11-08 01:12:10 |
```

```
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'))—  
2020-11-03 22:33:19 | 2020-11-08 01:33:19 |
```

You can work with dates without using `INTERVAL`, just by adding or subtracting seconds, minutes, and hours. For example, an interval of one day can be set by adding `60*60*24`.

## Note

The `INTERVAL` syntax or `addDays` function are always preferred. Simple addition or subtraction (syntax like `now() + ...`) doesn't consider time settings. For example, daylight saving time.

Examples:

```
SELECT toDateTime('2014-10-26 00:00:00', 'Europe/Moscow') AS time, time + 60 * 60 * 24 AS time_plus_24_hours,  
time + toIntervalDay(1) AS time_plus_1_day;
```

```
time time_plus_24_hours time_plus_1_day—  
2014-10-26 00:00:00 | 2014-10-26 23:00:00 | 2014-10-27 00:00:00 |
```

## See Also

- [Interval](#) data type
- [toInterval](#) type conversion functions

## Logical AND Operator

Syntax `SELECT a AND b` — calculates logical conjunction of `a` and `b` with the function `and`.

## Logical OR Operator

Syntax `SELECT a OR b` — calculates logical disjunction of `a` and `b` with the function `or`.

## Logical Negation Operator

Syntax `SELECT NOT a` — calculates logical negation of `a` with the function `not`.

## Conditional Operator

`a ? b : c` – The `if(a, b, c)` function.

Note:

The conditional operator calculates the values of `b` and `c`, then checks whether condition `a` is met, and then returns the corresponding value. If `b` or `c` is an `arrayJoin()` function, each row will be replicated regardless of the “`a`” condition.

## Conditional Expression

```
CASE [x]
  WHEN a THEN b
  [WHEN ... THEN ...]
  [ELSE c]
END
```

If `x` is specified, then `transform(x, [a, ...], [b, ...], c)` function is used. Otherwise – `multilf(a, b, ..., c)`.

If there is no `ELSE c` clause in the expression, the default value is `NULL`.

The `transform` function does not work with `NULL`.

## Concatenation Operator

`s1 || s2` – The `concat(s1, s2)` function.

## Lambda Creation Operator

`x -> expr` – The `lambda(x, expr)` function.

The following operators do not have a priority since they are brackets:

## Array Creation Operator

`[x1, ...]` – The `array(x1, ...)` function.

## Tuple Creation Operator

`(x1, x2, ...)` – The `tuple(x2, x2, ...)` function.

## Associativity

All binary operators have left associativity. For example, `1 + 2 + 3` is transformed to `plus(plus(1, 2), 3)`. Sometimes this does not work the way you expect. For example, `SELECT 4 > 2 > 3` will result in 0.

For efficiency, the `and` and `or` functions accept any number of arguments. The corresponding chains of `AND` and `OR` operators are transformed into a single call of these functions.

## Checking for NULL

ClickHouse supports the `IS NULL` and `IS NOT NULL` operators.

### IS NULL

- For `Nullable` type values, the `IS NULL` operator returns:
  - 1, if the value is `NULL`.
  - 0 otherwise.
- For other values, the `IS NULL` operator always returns 0.

Can be optimized by enabling the `optimize_functions_to_subcolumns` setting. With `optimize_functions_to_subcolumns = 1` the function reads only `null` subcolumn instead of reading and processing the whole column data. The query `SELECT n IS NULL FROM table` transforms to `SELECT n.null FROM TABLE`.

```
SELECT x+100 FROM t_null WHERE y IS NULL
```

```
plus(x, 100) —  
 101 |
```

## IS NOT NULL

- For `Nullable` type values, the `IS NOT NULL` operator returns:
  - `0`, if the value is `NULL`.
  - `1` otherwise.
- For other values, the `IS NOT NULL` operator always returns `1`.

```
SELECT * FROM t_null WHERE y IS NOT NULL
```

```
x — y  
2 | 3 |
```

Can be optimized by enabling the `optimize_functions_to_subcolumns` setting. With `optimize_functions_to_subcolumns = 1` the function reads only `null` subcolumn instead of reading and processing the whole column data. The query `SELECT n IS NOT NULL FROM table` transforms to `SELECT NOT n.null FROM TABLE`.

## 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.

ClickHouse allows types to differ in the left and the right parts of `IN` subquery. In this case it converts the left side value to the type of the right side, as if the `accurateCastOrNull` function is applied. That means, that the data type becomes `Nullable`, and if the conversion cannot be performed, it returns `NULL`.

## Example

Query:

```
SELECT '1' IN (SELECT 1);
```

Result:

```
in('1', _subquery49)
  1 |
```

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` always equals 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 if `transform_null_in = 0`.

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 = 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

```
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 do not 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 do not 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.

## Distributed Subqueries and `max_parallel_replicas`

When `max_parallel_replicas` is greater than 1, distributed queries are further transformed. For example, the following:

```
SELECT CounterID, count() FROM distributed_table_1 WHERE UserID IN (SELECT UserID FROM local_table_2 WHERE  
CounterID < 100)  
SETTINGS max_parallel_replicas=3
```

is transformed on each server into

```
SELECT CounterID, count() FROM local_table_1 WHERE UserID IN (SELECT UserID FROM local_table_2 WHERE  
CounterID < 100)  
SETTINGS parallel_replicas_count=3, parallel_replicas_offset=M
```

where M is between 1 and 3 depending on which replica the local query is executing on. These settings affect every MergeTree-family table in the query and have the same effect as applying SAMPLE 1/3 OFFSET (M-1)/3 on each table.

Therefore adding the max\_parallel\_replicas setting will only produce correct results if both tables have the same replication scheme and are sampled by UserID or a subkey of it. In particular, if local\_table\_2 does not have a sampling key, incorrect results will be produced. The same rule applies to JOIN.

One workaround if local\_table\_2 does not meet the requirements, is to use GLOBAL IN or GLOBAL JOIN.

## ANSI SQL Compatibility of ClickHouse SQL Dialect

### Note

This article relies on Table 38, “Feature taxonomy and definition for mandatory features”, Annex F of [ISO/IEC CD 9075-2:2011](#).

## Differences in Behaviour

The following table lists cases when query feature works in ClickHouse, but behaves not as specified in ANSI SQL.

Feature ID	Feature Name	Difference
E011	Numeric data types	Numeric literal with period is interpreted as approximate (Float64) instead of exact (Decimal)
E051-05	Select items can be renamed	Item renames have a wider visibility scope than just the SELECT result
E141-01	NOT NULL constraints	NOT NULL is implied for table columns by default
E011-04	Arithmetic operators	ClickHouse overflows instead of checked arithmetic and changes the result data type based on custom rules

## Feature Status

Feature ID	Feature Name	Status	Comment
E011	Numeric data types	Partial	
E011-01	INTEGER and SMALLINT data types	Yes	
E011-02	REAL, DOUBLE PRECISION and FLOAT data types	Yes	
E011-03	DECIMAL and NUMERIC data types	Yes	
E011-04	Arithmetic operators	Yes	

Feature ID	Feature Name	Status	Comment
E011-05	Numeric comparison	Yes	
E011-06	Implicit casting among the numeric data types	No	ANSI SQL allows arbitrary implicit cast between numeric types, while ClickHouse relies on functions having multiple overloads instead of implicit cast
<b>E021</b>	<b>Character string types</b>	<b>Partial</b>	
E021-01	CHARACTER data type	Yes	
E021-02	CHARACTER VARYING data type	Yes	
E021-03	Character literals	Yes	
E021-04	CHARACTER_LENGTH function	Partial	No USING clause
E021-05	OCTET_LENGTH function	No	LENGTH behaves similarly
E021-06	SUBSTRING	Partial	No support for SIMILAR and ESCAPE clauses, no SUBSTRING_REGEX variant
E021-07	Character concatenation	Partial	No COLLATE clause
E021-08	UPPER and LOWER functions	Yes	
E021-09	TRIM function	Yes	
E021-10	Implicit casting among the fixed-length and variable-length character string types	Partial	ANSI SQL allows arbitrary implicit cast between string types, while ClickHouse relies on functions having multiple overloads instead of implicit cast
E021-11	POSITION function	Partial	No support for IN and USING clauses, no POSITION_REGEX variant
E021-12	Character comparison	Yes	
<b>E031</b>	<b>Identifiers</b>	<b>Partial</b>	
E031-01	Delimited identifiers	Partial	Unicode literal support is limited
E031-02	Lower case identifiers	Yes	
E031-03	Trailing underscore	Yes	
<b>E051</b>	<b>Basic query specification</b>	<b>Partial</b>	
E051-01	SELECT DISTINCT	Yes	

Feature ID	Feature Name	Status	Comment
E051-02	GROUP BY clause	Yes	
E051-04	GROUP BY can contain columns not in <select list>	Yes	
E051-05	Select items can be renamed	Yes	
E051-06	HAVING clause	Yes	
E051-07	Qualified * in select list	Yes	
E051-08	Correlation name in the FROM clause	Yes	
E051-09	Rename columns in the FROM clause	No	
<b>E061</b>	<b>Basic predicates and search conditions</b>	<b>Partial</b>	
E061-01	Comparison predicate	Yes	
E061-02	BETWEEN predicate	Partial	No SYMMETRIC and ASYMMETRIC clause
E061-03	IN predicate with list of values	Yes	
E061-04	LIKE predicate	Yes	
E061-05	LIKE predicate: ESCAPE clause	No	
E061-06	NULL predicate	Yes	
E061-07	Quantified comparison predicate	No	
E061-08	EXISTS predicate	No	
E061-09	Subqueries in comparison predicate	Yes	
E061-11	Subqueries in IN predicate	Yes	
E061-12	Subqueries in quantified comparison predicate	No	
E061-13	Correlated subqueries	No	
E061-14	Search condition	Yes	
<b>E071</b>	<b>Basic query expressions</b>	<b>Partial</b>	

Feature ID	Feature Name	Status	Comment
E071-01	UNION DISTINCT table operator	Yes	
E071-02	UNION ALL table operator	Yes	
E071-03	EXCEPT DISTINCT table operator	No	
E071-05	Columns combined via table operators need not have exactly the same data type	Yes	
E071-06	Table operators in subqueries	Yes	
<b>E081</b>	<b>Basic privileges</b>	<b>Yes</b>	
E081-01	SELECT privilege at the table level	Yes	
E081-02	DELETE privilege		
E081-03	INSERT privilege at the table level	Yes	
E081-04	UPDATE privilege at the table level	Yes	
E081-05	UPDATE privilege at the column level		
E081-06	REFERENCES privilege at the table level		
E081-07	REFERENCES privilege at the column level		
E081-08	WITH GRANT OPTION	Yes	
E081-09	USAGE privilege		
E081-10	EXECUTE privilege		
<b>E091</b>	<b>Set functions</b>	<b>Yes</b>	
E091-01	AVG	Yes	
E091-02	COUNT	Yes	
E091-03	MAX	Yes	
E091-04	MIN	Yes	
E091-05	SUM	Yes	
E091-06	ALL quantifier	Yes	

Feature ID	Feature Name	Status	Comment
E091-07	DISTINCT quantifier	Yes	Not all aggregate functions supported
<b>E101</b>	<b>Basic data manipulation</b>	Partial	
E101-01	INSERT statement	Yes	Note: primary key in ClickHouse does not imply the UNIQUE constraint
E101-03	Searched UPDATE statement	Partial	There's an <code>ALTER UPDATE</code> statement for batch data modification
E101-04	Searched DELETE statement	Partial	There's an <code>ALTER DELETE</code> statement for batch data removal
<b>E111</b>	<b>Single row SELECT statement</b>	No	
<b>E121</b>	<b>Basic cursor support</b>	No	
E121-01	DECLARE CURSOR	No	
E121-02	ORDER BY columns need not be in select list	Yes	
E121-03	Value expressions in ORDER BY clause	Yes	
E121-04	OPEN statement	No	
E121-06	Positioned UPDATE statement	No	
E121-07	Positioned DELETE statement	No	
E121-08	CLOSE statement	No	
E121-10	FETCH statement: implicit NEXT	No	
E121-17	WITH HOLD cursors	No	
<b>E131</b>	<b>Null value support (nulls in lieu of values)</b>	Yes	Some restrictions apply
<b>E141</b>	<b>Basic integrity constraints</b>	Partial	
E141-01	NOT NULL constraints	Yes	Note: NOT NULL is implied for table columns by default
E141-02	UNIQUE constraint of NOT NULL columns	No	
E141-03	PRIMARY KEY constraints	Partial	

Feature ID	Feature Name	Status	Comment
E141-04	Basic FOREIGN KEY constraint with the NO ACTION default for both referential delete action and referential update action	No	
E141-06	CHECK constraint	Yes	
E141-07	Column defaults	Yes	
E141-08	NOT NULL inferred on PRIMARY KEY	Yes	
E141-10	Names in a foreign key can be specified in any order	No	
<b>E151</b>	<b>Transaction support</b>	No	
E151-01	COMMIT statement	No	
E151-02	ROLLBACK statement	No	
<b>E152</b>	<b>Basic SET TRANSACTION statement</b>	No	
E152-01	SET TRANSACTION statement: ISOLATION LEVEL SERIALIZABLE clause	No	
E152-02	SET TRANSACTION statement: READ ONLY and READ WRITE clauses	No	
<b>E153</b>	<b>Updatable queries with subqueries</b>	Yes	
<b>E161</b>	<b>SQL comments using leading double minus</b>	Yes	
<b>E171</b>	<b>SQLSTATE support</b>	No	
<b>E182</b>	<b>Host language binding</b>	No	
<b>F031</b>	<b>Basic schema manipulation</b>	Partial	
F031-01	CREATE TABLE statement to create persistent base tables	Partial	No SYSTEM VERSIONING, ON COMMIT, GLOBAL, LOCAL, PRESERVE, DELETE, REF IS, WITH OPTIONS, UNDER, LIKE, PERIOD FOR clauses and no support for user resolved data types

Feature ID	Feature Name	Status	Comment
F031-02	CREATE VIEW statement	Partial	No RECURSIVE, CHECK, UNDER, WITH OPTIONS clauses and no support for user resolved data types
F031-03	GRANT statement	Yes	
F031-04	ALTER TABLE statement: ADD COLUMN clause	Yes	No support for GENERATED clause and system time period
F031-13	DROP TABLE statement: RESTRICT clause	No	
F031-16	DROP VIEW statement: RESTRICT clause	No	
F031-19	REVOKE statement: RESTRICT clause	No	
<b>F041</b>	<b>Basic joined table</b>	Partial	
F041-01	Inner join (but not necessarily the INNER keyword)	Yes	
F041-02	INNER keyword	Yes	
F041-03	LEFT OUTER JOIN	Yes	
F041-04	RIGHT OUTER JOIN	Yes	
F041-05	Outer joins can be nested	Yes	
F041-07	The inner table in a left or right outer join can also be used in an inner join	Yes	
F041-08	All comparison operators are supported (rather than just =)	No	
<b>F051</b>	<b>Basic date and time</b>	Partial	
F051-01	DATE data type (including support of DATE literal)	Yes	
F051-02	TIME data type (including support of TIME literal) with fractional seconds precision of at least 0	No	

Feature ID	Feature Name	Status	Comment
F051-03	TIMESTAMP data type (including support of TIMESTAMP literal) with fractional seconds precision of at least 0 and 6	Yes	
F051-04	Comparison predicate on DATE, TIME, and TIMESTAMP data types	Yes	
F051-05	Explicit CAST between datetime types and character string types	Yes	
F051-06	CURRENT_DATE	No	today() is similar
F051-07	LOCALTIME	No	now() is similar
F051-08	LOCALTIMESTAMP	No	
<b>F081</b>	<b>UNION and EXCEPT in views</b>	Partial	
<b>F131</b>	<b>Grouped operations</b>	Partial	
F131-01	WHERE, GROUP BY, and HAVING clauses supported in queries with grouped views	Yes	
F131-02	Multiple tables supported in queries with grouped views	Yes	
F131-03	Set functions supported in queries with grouped views	Yes	
F131-04	Subqueries with GROUP BY and HAVING clauses and grouped views	Yes	
F131-05	Single row SELECT with GROUP BY and HAVING clauses and grouped views	No	
<b>F181</b>	<b>Multiple module support</b>	No	
<b>F201</b>	<b>CAST function</b>	Yes	
<b>F221</b>	<b>Explicit defaults</b>	No	
<b>F261</b>	<b>CASE expression</b>	Yes	
F261-01	Simple CASE	Yes	
F261-02	Searched CASE	Yes	

Feature ID	Feature Name	Status	Comment
F261-03	NULLIF	Yes	
F261-04	COALESCE	Yes	
<b>F311</b>	<b>Schema definition statement</b>	<b>Partial</b>	
F311-01	CREATE SCHEMA	Partial	See CREATE DATABASE
F311-02	CREATE TABLE for persistent base tables	Yes	
F311-03	CREATE VIEW	Yes	
F311-04	CREATE VIEW: WITH CHECK OPTION	No	
F311-05	GRANT statement	Yes	
<b>F471</b>	<b>Scalar subquery values</b>	<b>Yes</b>	
<b>F481</b>	<b>Expanded NULL predicate</b>	<b>Yes</b>	
<b>F812</b>	<b>Basic flagging</b>	<b>No</b>	
<b>S011</b>	<b>Distinct data types</b>		
<b>T321</b>	<b>Basic SQL-invoked routines</b>	<b>No</b>	
T321-01	User-defined functions with no overloading	No	
T321-02	User-defined stored procedures with no overloading	No	
T321-03	Function invocation	No	
T321-04	CALL statement	No	
T321-05	RETURN statement	No	
<b>T631</b>	<b>IN predicate with one list element</b>	<b>Yes</b>	

## [experimental] Window Functions

ClickHouse supports the standard grammar for defining windows and window functions. The following features are currently supported:

Feature	Support or workaround
ad hoc window specification <code>(count(*) over (partition by id order by time desc))</code>	supported
expressions involving window functions, e.g. <code>(count(*) over ()) / 2</code>	not supported, wrap in a subquery ( <a href="#">feature request</a> )
WINDOW clause (select ... from table window w as (partiton by id))	supported
ROWS frame	supported
RANGE frame	supported, the default
INTERVAL syntax for DateTime RANGE OFFSET frame	not supported, specify the number of seconds instead
GROUPS frame	not supported
Calculating aggregate functions over a frame <code>(sum(value) over (order by time))</code>	all aggregate functions are supported
<code>rank()</code> , <code>dense_rank()</code> , <code>row_number()</code>	supported
<code>lag/lead(value, offset)</code>	Not supported. Workarounds:  1) replace with <code>any(value) over (.... rows between &lt;offset&gt; preceding and &lt;offset&gt; preceding)</code> , or following for <code>lead</code>  2) use <code>lagInFrame/leadInFrame</code> , which are analogous, but respect the window frame. To get behavior identical to <code>lag/lead</code> , use <code>rows between unbounded preceding and unbounded following</code>

## References

### GitHub Issues

The roadmap for the initial support of window functions is [in this issue](#).

All GitHub issues related to window funtions have the [comp-window-functions](#) tag.

### Tests

These tests contain the examples of the currently supported grammar:

[https://github.com/ClickHouse/ClickHouse/blob/master/tests/performance/window\\_functions.xml](https://github.com/ClickHouse/ClickHouse/blob/master/tests/performance/window_functions.xml)

[https://github.com/ClickHouse/ClickHouse/blob/master/tests/queries/0\\_stateless/01591\\_window\\_functions.sql](https://github.com/ClickHouse/ClickHouse/blob/master/tests/queries/0_stateless/01591_window_functions.sql)

## Postgres Docs

<https://www.postgresql.org/docs/current/sql-select.html#SQL-WINDOW>

<https://www.postgresql.org/docs/devel/sql-expressions.html#SYNTAX-WINDOW-FUNCTIONS>

<https://www.postgresql.org/docs/devel/functions-window.html>

<https://www.postgresql.org/docs/devel/tutorial-window.html>

## MySQL Docs

<https://dev.mysql.com/doc/refman/8.0/en/window-function-descriptions.html>

<https://dev.mysql.com/doc/refman/8.0/en/window-functions-usage.html>

<https://dev.mysql.com/doc/refman/8.0/en/window-functions-frames.html>

---

## ClickHouse Guides

List of detailed step-by-step instructions that help to solve various tasks using ClickHouse:

- [Tutorial on simple cluster set-up](#)
  - [Applying a CatBoost model in ClickHouse](#)
- 

## Applying a Catboost Model in ClickHouse

**CatBoost** is a free and open-source gradient boosting library developed at [Yandex](#) for machine learning.

With this instruction, you will learn to apply pre-trained models in ClickHouse by running model inference from SQL.

To apply a CatBoost model in ClickHouse:

1. [Create a Table](#).
2. [Insert the Data to the Table](#).
3. [Integrate CatBoost into ClickHouse](#) (Optional step).
4. [Run the Model Inference from SQL](#).

For more information about training CatBoost models, see [Training and applying models](#).

You can reload CatBoost models if the configuration was updated without restarting the server using `RELOAD MODEL` and `RELOAD MODELS` system queries.

## Prerequisites

If you do not have the **Docker** yet, install it.

### Note

**Docker** is a software platform that allows you to create containers that isolate a CatBoost and ClickHouse installation from the rest of the system.

Before applying a CatBoost model:

1. Pull the **Docker image** from the registry:

```
$ docker pull yandex/tutorial-catboost-clickhouse
```

This Docker image contains everything you need to run CatBoost and ClickHouse: code, runtime, libraries, environment variables, and configuration files.

**2.** Make sure the Docker image has been successfully pulled:

```
$ docker image ls
REPOSITORY           TAG      IMAGE ID      CREATED       SIZE
yandex/tutorial-catboost-clickhouse  latest   622e4d17945b  22 hours ago  1.37GB
```

**3.** Start a Docker container based on this image:

```
$ docker run -it -p 8888:8888 yandex/tutorial-catboost-clickhouse
```

## 1. Create a Table

To create a ClickHouse table for the training sample:

**1.** Start ClickHouse console client in the interactive mode:

```
$ clickhouse client
```

### Note

The ClickHouse server is already running inside the Docker container.

**2.** Create the table using the command:

```
:) 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_CODE UInt32
)
ENGINE = MergeTree ORDER BY date
```

**3.** Exit from ClickHouse console client:

```
:) exit
```

## 2. Insert the Data to the Table

To insert the data:

**1.** Run the following command:

```
$ clickhouse client --host 127.0.0.1 --query 'INSERT INTO amazon_train FORMAT CSVWithNames' < ~/amazon/train.csv
```

## 2. Start ClickHouse console client in the interactive mode:

```
$ clickhouse client
```

## 3. Make sure the data has been uploaded:

```
:) SELECT count() FROM amazon_train  
  
SELECT count()  
FROM amazon_train  
  
+-count()-+  
| 65538 |  
+-----+
```

## 3. Integrate CatBoost into ClickHouse

### Note

**Optional step.** The Docker image contains everything you need to run CatBoost and ClickHouse.

To integrate CatBoost into ClickHouse:

### 1. Build the evaluation library.

The fastest way to evaluate a CatBoost model is compile `libcatboostmodel.<so|dll|dylib>` library. For more information about how to build the library, see [CatBoost documentation](#).

### 2. Create a new directory anywhere and with any name, for example, `data` and put the created library in it. The Docker image already contains the library `data/libcatboostmodel.so`.

### 3. Create a new directory for config model anywhere and with any name, for example, `models`.

### 4. Create a model configuration file with any name, for example, `models/amazon_model.xml`.

### 5. Describe the model configuration:

```
<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. Add the path to CatBoost and the model configuration to the ClickHouse configuration:

```
<!-- 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>
```

## Note

You can change path to the CatBoost model configuration later without restarting server.

## 4. Run the Model Inference from SQL

For test model run the ClickHouse client `$ clickhouse client`.

Let's make sure that the model is working:

```
:) 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
```

## Note

Function **modelEvaluate** returns tuple with per-class raw predictions for multiclass models.

Let's predict the probability:

```
:) 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
```

## Note

More info about **exp()** function.

Let's calculate LogLoss on the sample:

```
:) 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
)
```

## Note

More info about **avg()** and **log()** functions.

# Operations

ClickHouse operations manual consists of the following major sections:

- Requirements
- Monitoring
- Troubleshooting
- Usage Recommendations
- Update Procedure
- Access Rights
- Data Backup
- Configuration Files
- Quotas
- System Tables
- Server Configuration Parameters
- How To Test Your Hardware With ClickHouse
- Settings
- Utilities

## Requirements

### CPU

For installation from prebuilt deb packages, use a CPU with x86\_64 architecture and support for SSE 4.2 instructions. To run ClickHouse with processors that do not support SSE 4.2 or have AArch64 or PowerPC64LE architecture, you should build ClickHouse from sources.

ClickHouse implements parallel data processing and uses all the hardware resources available. When choosing a processor, take into account that ClickHouse works more efficiently at configurations with a large number of cores but a lower clock rate than at configurations with fewer cores and a higher clock rate. For example, 16 cores with 2600 MHz is preferable to 8 cores with 3600 MHz.

It is recommended to use **Turbo Boost** and **hyper-threading** technologies. It significantly improves performance with a typical workload.

## RAM

We recommend using a minimum of 4GB of RAM to perform non-trivial queries. The ClickHouse server can run with a much smaller amount of RAM, but it requires memory for processing queries.

The required volume of RAM depends on:

- The complexity of queries.
- The amount of data that is processed in queries.

To calculate the required volume of RAM, you should estimate the size of temporary data for **GROUP BY**, **DISTINCT**, **JOIN** and other operations you use.

ClickHouse can use external memory for temporary data. See [GROUP BY in External Memory](#) for details.

## Swap File

Disable the swap file for production environments.

## Storage Subsystem

You need to have 2GB of free disk space to install ClickHouse.

The volume of storage required for your data should be calculated separately. Assessment should include:

- Estimation of the data volume.

You can take a sample of the data and get the average size of a row from it. Then multiply the value by the number of rows you plan to store.

- The data compression coefficient.

To estimate the data compression coefficient, load a sample of your data into ClickHouse, and compare the actual size of the data with the size of the table stored. For example, clickstream data is usually compressed by 6-10 times.

To calculate the final volume of data to be stored, apply the compression coefficient to the estimated data volume. If you plan to store data in several replicas, then multiply the estimated volume by the number of replicas.

## Network

If possible, use networks of 10G or higher class.

The network bandwidth is critical for processing distributed queries with a large amount of intermediate data. Besides, network speed affects replication processes.

# Software

ClickHouse is developed primarily for the Linux family of operating systems. The recommended Linux distribution is Ubuntu. The `tzdata` package should be installed in the system.

ClickHouse can also work in other operating system families. See details in the [Getting started](#) section of the documentation.

---

# Monitoring

You can monitor:

- Utilization of hardware resources.
- ClickHouse server metrics.

## Resource Utilization

ClickHouse does not monitor the state of hardware resources by itself.

It is highly recommended to set up monitoring for:

- Load and temperature on processors.  
You can use [dmesg](#), [turbostat](#) or other instruments.
- Utilization of storage system, RAM and network.

## ClickHouse Server Metrics

ClickHouse server has embedded instruments for self-state monitoring.

To track server events use server logs. See the [logger](#) section of the configuration file.

ClickHouse collects:

- Different metrics of how the server uses computational resources.
- Common statistics on query processing.

You can find metrics in the [system.metrics](#), [system.events](#), and [system.asynchronous\\_metrics](#) tables.

You can configure ClickHouse to export metrics to [Graphite](#). See the [Graphite section](#) in the ClickHouse server configuration file. Before configuring export of metrics, you should set up Graphite by following their official [guide](#).

You can configure ClickHouse to export metrics to [Prometheus](#). See the [Prometheus section](#) in the ClickHouse server configuration file. Before configuring export of metrics, you should set up Prometheus by following their official [guide](#).

Additionally, you can monitor server availability through the HTTP API. Send the `HTTP GET` request to `/ping`. If the server is available, it responds with `200 OK`.

To monitor servers in a cluster configuration, you should set the [max\\_replica\\_delay\\_for\\_distributed\\_queries](#) parameter and use the HTTP resource `/replicas_status`. A request to `/replicas_status` returns `200 OK` if the replica is available and is not delayed behind the other replicas. If a replica is delayed, it returns `503 HTTP_SERVICE_UNAVAILABLE` with information about the gap.

---

# Troubleshooting

- Installation
- Connecting to the server
- Query processing
- Efficiency of query processing

## Installation

### You Cannot Get Deb Packages from ClickHouse Repository with Apt-get

- Check firewall settings.
- If you cannot access the repository for any reason, download packages as described in the [Getting started](#) article and install them manually using the `sudo dpkg -i <packages>` command. You will also need the `tzdata` package.

## Connecting to the Server

Possible issues:

- The server is not running.
- Unexpected or wrong configuration parameters.

## Server Is Not Running

### Check if server is running

Command:

```
$ sudo service clickhouse-server status
```

If the server is not running, start it with the command:

```
$ sudo service clickhouse-server start
```

### Check logs

The main log of `clickhouse-server` is in `/var/log/clickhouse-server/clickhouse-server.log` by default.

If the server started successfully, you should see the strings:

- `<Information> Application: starting up.` — Server started.
- `<Information> Application: Ready for connections.` — Server is running and ready for connections.

If `clickhouse-server start` failed with a configuration error, you should see the `<Error>` string with an error description. For example:

```
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
```

If you do not see an error at the end of the file, look through the entire file starting from the string:

```
<Information> Application: starting up.
```

If you try to start a second instance of `clickhouse-server` on the server, you see the following log:

```
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
```

## See `system.d` logs

If you do not find any useful information in `clickhouse-server` logs or there aren't any logs, you can view `system.d` logs using the command:

```
$ sudo journalctl -u clickhouse-server
```

## Start `clickhouse-server` in interactive mode

```
$ sudo -u clickhouse /usr/bin/clickhouse-server --config-file /etc/clickhouse-server/config.xml
```

This command starts the server as an interactive app with standard parameters of the autostart script. In this mode `clickhouse-server` prints all the event messages in the console.

## Configuration Parameters

Check:

- Docker settings.

If you run ClickHouse in Docker in an IPv6 network, make sure that `network=host` is set.

- Endpoint settings.

Check `listen_host` and `tcp_port` settings.

ClickHouse server accepts localhost connections only by default.

- HTTP protocol settings.

Check protocol settings for the HTTP API.

- Secure connection settings.

Check:

- The `tcp_port_secure` setting.

- Settings for [SSL certificates](#).

Use proper parameters while connecting. For example, use the `port_secure` parameter with `clickhouse_client`.

- User settings.

You might be using the wrong user name or password.

## Query Processing

If ClickHouse is not able to process the query, it sends an error description to the client. In the `clickhouse-client` you get a description of the error in the console. If you are using the HTTP interface, ClickHouse sends the error description in the response body. For example:

```
$ 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
```

If you start `clickhouse-client` with the `stack-trace` parameter, ClickHouse returns the server stack trace with the description of an error.

You might see a message about a broken connection. In this case, you can repeat the query. If the connection breaks every time you perform the query, check the server logs for errors.

## Efficiency of Query Processing

If you see that ClickHouse is working too slowly, you need to profile the load on the server resources and network for your queries.

You can use the `clickhouse-benchmark` utility to profile queries. It shows the number of queries processed per second, the number of rows processed per second, and percentiles of query processing times.

## ClickHouse Upgrade

If ClickHouse was installed from `deb` packages, execute the following commands on the server:

```
$ sudo apt-get update
$ sudo apt-get install clickhouse-client clickhouse-server
$ sudo service clickhouse-server restart
```

If you installed ClickHouse using something other than the recommended `deb` packages, use the appropriate update method.

### Note

You can update multiple servers at once as soon as there is no moment when all replicas of one shard are offline.

The upgrade of older version of ClickHouse to specific version:

As an example:

`xx.yy.a.b` is a current stable version. The latest stable version could be found [here](#)

```
$ sudo apt-get update
$ sudo apt-get install clickhouse-server=xx.yy.a.b clickhouse-client=xx.yy.a.b clickhouse-common-static=xx.yy.a.b
$ sudo service clickhouse-server restart
```

# Access Control and Account Management

ClickHouse supports access control management based on **RBAC** approach.

ClickHouse access entities:

- **User account**
- **Role**
- **Row Policy**
- **Settings Profile**
- **Quota**

You can configure access entities using:

- SQL-driven workflow.
  - You need to **enable** this functionality.
- Server configuration files `users.xml` and `config.xml`.

We recommend using SQL-driven workflow. Both of the configuration methods work simultaneously, so if you use the server configuration files for managing accounts and access rights, you can smoothly switch to SQL-driven workflow.

## Warning

You can't manage the same access entity by both configuration methods simultaneously.

To see all users, roles, profiles, etc. and all their grants use **SHOW ACCESS** statement.

## Usage

By default, the ClickHouse server provides the `default` user account which is not allowed using SQL-driven access control and account management but has all the rights and permissions. The `default` user account is used in any cases when the username is not defined, for example, at login from client or in distributed queries. In distributed query processing a default user account is used, if the configuration of the server or cluster does not specify the **user and password** properties.

If you just started using ClickHouse, consider the following scenario:

1. **Enable** SQL-driven access control and account management for the `default` user.
2. Log in to the `default` user account and create all the required users. Don't forget to create an administrator account (`GRANT ALL ON *.* TO admin_user_account WITH GRANT OPTION`).
3. **Restrict permissions** for the `default` user and disable SQL-driven access control and account management for it.

## Properties of Current Solution

- You can grant permissions for databases and tables even if they do not exist.
- If a table was deleted, all the privileges that correspond to this table are not revoked. This means that even if you create a new table with the same name later, all the privileges remain valid. To revoke privileges corresponding to the deleted table, you need to execute, for example, the `REVOKE ALL PRIVILEGES ON db.table FROM ALL` query.
- There are no lifetime settings for privileges.

## User Account

A user account is an access entity that allows to authorize someone in ClickHouse. A user account contains:

- Identification information.
- **Privileges** that define a scope of queries the user can execute.
- Hosts allowed to connect to the ClickHouse server.
- Assigned and default roles.
- Settings with their constraints applied by default at user login.
- Assigned settings profiles.

Privileges can be granted to a user account by the **GRANT** query or by assigning **roles**. To revoke privileges from a user, ClickHouse provides the **REVOKE** query. To list privileges for a user, use the **SHOW GRANTS** statement.

Management queries:

- **CREATE USER**
- **ALTER USER**
- **DROP USER**
- **SHOW CREATE USER**
- **SHOW USERS**

## Settings Applying

Settings can be configured differently: for a user account, in its granted roles and in settings profiles. At user login, if a setting is configured for different access entities, the value and constraints of this setting are applied as follows (from higher to lower priority):

1. User account settings.
2. The settings of default roles of the user account. If a setting is configured in some roles, then order of the setting application is undefined.
3. The settings from settings profiles assigned to a user or to its default roles. If a setting is configured in some profiles, then order of setting application is undefined.
4. Settings applied to all the server by default or from the **default profile**.

## Role

Role is a container for access entities that can be granted to a user account.

Role contains:

- **Privileges**
- Settings and constraints
- List of assigned roles

Management queries:

- **CREATE ROLE**
- **ALTER ROLE**

- [DROP ROLE](#)
- [SET ROLE](#)
- [SET DEFAULT ROLE](#)
- [SHOW CREATE ROLE](#)
- [SHOW ROLES](#)

Privileges can be granted to a role by the [GRANT](#) query. To revoke privileges from a role ClickHouse provides the [REVOKE](#) query.

## Row Policy

Row policy is a filter that defines which of the rows are available to a user or a role. Row policy contains filters for one particular table, as well as a list of roles and/or users which should use this row policy.

### Warning

Row policies makes sense only for users with readonly access. If user can modify table or copy partitions between tables, it defeats the restrictions of row policies.

Management queries:

- [CREATE ROW POLICY](#)
- [ALTER ROW POLICY](#)
- [DROP ROW POLICY](#)
- [SHOW CREATE ROW POLICY](#)
- [SHOW POLICIES](#)

## Settings Profile

Settings profile is a collection of [settings](#). Settings profile contains settings and constraints, as well as a list of roles and/or users to which this profile is applied.

Management queries:

- [CREATE SETTINGS PROFILE](#)
- [ALTER SETTINGS PROFILE](#)
- [DROP SETTINGS PROFILE](#)
- [SHOW CREATE SETTINGS PROFILE](#)
- [SHOW PROFILES](#)

## Quota

Quota limits resource usage. See [Quotas](#).

Quota contains a set of limits for some durations, as well as a list of roles and/or users which should use this quota.

Management queries:

- [CREATE QUOTA](#)
- [ALTER QUOTA](#)
- [DROP QUOTA](#)
- [SHOW CREATE QUOTA](#)
- [SHOW QUOTA](#)
- [SHOW QUOTAS](#)

## Enabling SQL-driven Access Control and Account Management

- Setup a directory for configurations storage.

ClickHouse stores access entity configurations in the folder set in the [access\\_control\\_path](#) server configuration parameter.

- Enable SQL-driven access control and account management for at least one user account.

By default, SQL-driven access control and account management is disabled for all users. You need to configure at least one user in the [users.xml](#) configuration file and set the value of the [access\\_management](#) setting to 1.

---

## External User Authenticators and Directories

ClickHouse supports authenticating and managing users using external services.

The following external authenticators and directories are supported:

- [LDAP Authenticator](#) and [Directory](#)
- Kerberos [Authenticator](#)

---

## Kerberos

Existing and properly configured ClickHouse users can be authenticated via Kerberos authentication protocol.

Currently, Kerberos can only be used as an external authenticator for existing users, which are defined in [users.xml](#) or in local access control paths. Those users may only use HTTP requests and must be able to authenticate using GSS-SPNEGO mechanism.

For this approach, Kerberos must be configured in the system and must be enabled in ClickHouse config.

## Enabling Kerberos in ClickHouse

To enable Kerberos, one should include `kerberos` section in `config.xml`. This section may contain additional parameters.

Parameters:

- `principal` - canonical service principal name that will be acquired and used when accepting security contexts.
- This parameter is optional, if omitted, the default principal will be used.

- `realm` - a realm, that will be used to restrict authentication to only those requests whose initiator's realm matches it.
- This parameter is optional, if omitted, no additional filtering by realm will be applied.

Example (goes into `config.xml`):

```
<clickhouse>
  <!-- ... -->
  <kerberos />
</clickhouse>
```

With principal specification:

```
<clickhouse>
  <!-- ... -->
  <kerberos>
    <principal>HTTP/clickhouse.example.com@EXAMPLE.COM</principal>
  </kerberos>
</clickhouse>
```

With filtering by realm:

```
<clickhouse>
  <!-- ... -->
  <kerberos>
    <realm>EXAMPLE.COM</realm>
  </kerberos>
</clickhouse>
```

## Note

You can define only one kerberos section. The presence of multiple `kerberos` sections will force ClickHouse to disable Kerberos authentication.

## Note

`principal` and `realm` sections cannot be specified at the same time. The presence of both `principal` and `realm` sections will force ClickHouse to disable Kerberos authentication.

## Kerberos as an external authenticator for existing users

Kerberos can be used as a method for verifying the identity of locally defined users (users defined in `users.xml` or in local access control paths). Currently, **only** requests over the HTTP interface can be *kerberized* (via GSS-SPNEGO mechanism).

Kerberos principal name format usually follows this pattern:

- `primary/instance@REALM`

The `/instance` part may occur zero or more times. **The primary part of the canonical principal name of the initiator is expected to match the kerberized user name for authentication to succeed.**

## Enabling Kerberos in `users.xml`

In order to enable Kerberos authentication for the user, specify `kerberos` section instead of `password` or similar sections in the user definition.

Parameters:

- `realm` - a realm that will be used to restrict authentication to only those requests whose initiator's realm matches it.
- This parameter is optional, if omitted, no additional filtering by realm will be applied.

Example (goes into `users.xml`):

```
<clickhouse>
  <!-- ... -->
  <users>
    <!-- ... -->
    <my_user>
      <!-- ... -->
      <kerberos>
        <realm>EXAMPLE.COM</realm>
      </kerberos>
    </my_user>
  </users>
</clickhouse>
```

## Warning

Note that Kerberos authentication cannot be used alongside with any other authentication mechanism. The presence of any other sections like `password` alongside `kerberos` will force ClickHouse to shutdown.

## Reminder

Note, that now, once user `my_user` uses `kerberos`, Kerberos must be enabled in the main `config.xml` file as described previously.

## Enabling Kerberos using SQL

When [SQL-driven Access Control and Account Management](#) is enabled in ClickHouse, users identified by Kerberos can also be created using SQL statements.

```
CREATE USER my_user IDENTIFIED WITH kerberos REALM 'EXAMPLE.COM'
```

...or, without filtering by realm:

```
CREATE USER my_user IDENTIFIED WITH kerberos
```

## LDAP

LDAP server can be used to authenticate ClickHouse users. There are two different approaches for doing this:

- Use LDAP as an external authenticator for existing users, which are defined in `users.xml` or in local access control paths.

- Use LDAP as an external user directory and allow locally undefined users to be authenticated if they exist on the LDAP server.

For both of these approaches, an internally named LDAP server must be defined in the ClickHouse config so that other parts of the config can refer to it.

## LDAP Server Definition

To define LDAP server you must add `ldap_servers` section to the `config.xml`.

### Example

```
<clickhouse>
  <!-- ... -->
  <ldap_servers>
    <!-- Typical LDAP server. -->
    <my_ldap_server>
      <host>localhost</host>
      <port>636</port>
      <bind_dn>uid={user_name},ou=users,dc=example,dc=com</bind_dn>
      <verification_cooldown>300</verification_cooldown>
      <enable_tls>yes</enable_tls>
      <tls_minimum_protocol_version>tls1.2</tls_minimum_protocol_version>
      <tls_require_cert>demand</tls_require_cert>
      <tls_cert_file>/path/to/tls_cert_file</tls_cert_file>
      <tls_key_file>/path/to/tls_key_file</tls_key_file>
      <tls_ca_cert_file>/path/to/tls_ca_cert_file</tls_ca_cert_file>
      <tls_ca_cert_dir>/path/to/tls_ca_cert_dir</tls_ca_cert_dir>
      <tls_cipher_suite>ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:AES256-GCM-SHA384</tls_cipher_suite>
    </my_ldap_server>

    <!-- Typical Active Directory with configured user DN detection for further role mapping. -->
    <my_ad_server>
      <host>localhost</host>
      <port>389</port>
      <bind_dn>EXAMPLE\{user_name}</bind_dn>
      <user_dn_detection>
        <base_dn>CN=Users,DC=example,DC=com</base_dn>
        <search_filter>(&objectClass=user)(sAMAccountName={user_name})</search_filter>
      </user_dn_detection>
      <enable_tls>no</enable_tls>
    </my_ad_server>
  </ldap_servers>
</clickhouse>
```

Note, that you can define multiple LDAP servers inside the `ldap_servers` section using distinct names.

### Parameters

- `host` — LDAP server hostname or IP, this parameter is mandatory and cannot be empty.
- `port` — LDAP server port, default is `636` if `enable_tls` is set to `true`, `389` otherwise.
- `bind_dn` — Template used to construct the DN to bind to.
  - The resulting DN will be constructed by replacing all `{user_name}` substrings of the template with the actual user name during each authentication attempt.

- `user_dn_detection` — Section with LDAP search parameters for detecting the actual user DN of the bound user.
  - This is mainly used in search filters for further role mapping when the server is Active Directory. The resulting user DN will be used when replacing `{user_dn}` substrings wherever they are allowed. By default, user DN is set equal to bind DN, but once search is performed, it will be updated with to the actual detected user DN value.
  - `base_dn` — Template used to construct the base DN for the LDAP search.
    - The resulting DN will be constructed by replacing all `{user_name}` and `{bind_dn}` substrings of the template with the actual user name and bind DN during the LDAP search.
  - `scope` — Scope of the LDAP search.
    - Accepted values are: `base`, `one_level`, `children`, `subtree` (the default).
  - `search_filter` — Template used to construct the search filter for the LDAP search.
    - The resulting filter will be constructed by replacing all `{user_name}`, `{bind_dn}`, and `{base_dn}` substrings of the template with the actual user name, bind DN, and base DN during the LDAP search.
    - Note, that the special characters must be escaped properly in XML.
- `verification_cooldown` — A period of time, in seconds, after a successful bind attempt, during which the user will be assumed to be successfully authenticated for all consecutive requests without contacting the LDAP server.
  - Specify `0` (the default) to disable caching and force contacting the LDAP server for each authentication request.
- `enable_tls` — A flag to trigger the use of the secure connection to the LDAP server.
  - Specify `no` for plain text `ldap://` protocol (not recommended).
  - Specify `yes` for LDAP over SSL/TLS `ldaps://` protocol (recommended, the default).
  - Specify `starttls` for legacy StartTLS protocol (plain text `ldap://` protocol, upgraded to TLS).
- `tls_minimum_protocol_version` — The minimum protocol version of SSL/TLS.
  - Accepted values are: `ssl2`, `ssl3`, `tls1.0`, `tls1.1`, `tls1.2` (the default).
- `tls_require_cert` — SSL/TLS peer certificate verification behavior.
  - Accepted values are: `never`, `allow`, `try`, `demand` (the default).
- `tls_cert_file` — Path to certificate file.
- `tls_key_file` — Path to certificate key file.
- `tls_ca_cert_file` — Path to CA certificate file.
- `tls_ca_cert_dir` — Path to the directory containing CA certificates.
- `tls_cipher_suite` — Allowed cipher suite (in OpenSSL notation).

## LDAP External Authenticator

A remote LDAP server can be used as a method for verifying passwords for locally defined users (users defined in `users.xml` or in local access control paths). To achieve this, specify previously defined LDAP server name instead of `password` or similar sections in the user definition.

At each login attempt, ClickHouse tries to "bind" to the specified DN defined by the `bind_dn` parameter in the [LDAP server definition](#) using the provided credentials, and if successful, the user is considered authenticated. This is often called a "simple bind" method.

### Example

```
<clickhouse>
  <!-- ... -->
  <users>
    <!-- ... -->
    <my_user>
      <!-- ... -->
      <ldap>
        <server>my_ldap_server</server>
      </ldap>
    </my_user>
  </users>
</clickhouse>
```

Note, that user `my_user` refers to `my_ldap_server`. This LDAP server must be configured in the main `config.xml` file as described previously.

When SQL-driven [Access Control and Account Management](#) is enabled, users that are authenticated by LDAP servers can also be created using the [CREATE USER](#) statement.

Query:

```
CREATE USER my_user IDENTIFIED WITH ldap SERVER 'my_ldap_server';
```

## LDAP External User Directory

In addition to the locally defined users, a remote LDAP server can be used as a source of user definitions. To achieve this, specify previously defined LDAP server name (see [LDAP Server Definition](#)) in the `ldap` section inside the `users_directories` section of the `config.xml` file.

At each login attempt, ClickHouse tries to find the user definition locally and authenticate it as usual. If the user is not defined, ClickHouse will assume the definition exists in the external LDAP directory and will try to "bind" to the specified DN at the LDAP server using the provided credentials. If successful, the user will be considered existing and authenticated. The user will be assigned roles from the list specified in the `roles` section. Additionally, LDAP "search" can be performed and results can be transformed and treated as role names and then be assigned to the user if the `role_mapping` section is also configured. All this implies that the SQL-driven [Access Control and Account Management](#) is enabled and roles are created using the [CREATE ROLE](#) statement.

### Example

Goes into `config.xml`.

```

<clickhouse>
  <!-- ... -->
  <user_directories>
    <!-- Typical LDAP server. -->
    <ldap>
      <server>my_ldap_server</server>
      <roles>
        <my_local_role1 />
        <my_local_role2 />
      </roles>
      <role_mapping>
        <base_dn>ou=groups,dc=example,dc=com</base_dn>
        <scope>subtree</scope>
        <search_filter>(&(objectClass=groupOfNames)(member={bind_dn}))</search_filter>
        <attribute>cn</attribute>
        <prefix>clickhouse_</prefix>
      </role_mapping>
    </ldap>
    <!-- Typical Active Directory with role mapping that relies on the detected user DN. -->
    <ldap>
      <server>my_ad_server</server>
      <role_mapping>
        <base_dn>CN=Users,DC=example,DC=com</base_dn>
        <attribute>CN</attribute>
        <scope>subtree</scope>
        <search_filter>(&(objectClass=group)(member={user_dn}))</search_filter>
        <prefix>clickhouse_</prefix>
      </role_mapping>
    </ldap>
  </user_directories>
</clickhouse>

```

Note that `my_ldap_server` referred in the `ldap` section inside the `user_directories` section must be a previously defined LDAP server that is configured in the [config.xml](#) (see [LDAP Server Definition](#)).

## Parameters

- `server` — One of LDAP server names defined in the `ldap_servers` config section above. This parameter is mandatory and cannot be empty.
- `roles` — Section with a list of locally defined roles that will be assigned to each user retrieved from the LDAP server.
  - If no roles are specified here or assigned during role mapping (below), user will not be able to perform any actions after authentication.

- `role_mapping` — Section with LDAP search parameters and mapping rules.
  - When a user authenticates, while still bound to LDAP, an LDAP search is performed using `search_filter` and the name of the logged-in user. For each entry found during that search, the value of the specified attribute is extracted. For each attribute value that has the specified prefix, the prefix is removed, and the rest of the value becomes the name of a local role defined in ClickHouse, which is expected to be created beforehand by the `CREATE ROLE` statement.
  - There can be multiple `role_mapping` sections defined inside the same `ldap` section. All of them will be applied.
    - `base_dn` — Template used to construct the base DN for the LDAP search.
      - The resulting DN will be constructed by replacing all `{user_name}`, `{bind_dn}`, and `{user_dn}` substrings of the template with the actual user name, bind DN, and user DN during each LDAP search.
    - `scope` — Scope of the LDAP search.
      - Accepted values are: `base`, `one_level`, `children`, `subtree` (the default).
    - `search_filter` — Template used to construct the search filter for the LDAP search.
      - The resulting filter will be constructed by replacing all `{user_name}`, `{bind_dn}`, `{user_dn}`, and `{base_dn}` substrings of the template with the actual user name, bind DN, user DN, and base DN during each LDAP search.
    - Note, that the special characters must be escaped properly in XML.
    - `attribute` — Attribute name whose values will be returned by the LDAP search. `cn`, by default.
    - `prefix` — Prefix, that will be expected to be in front of each string in the original list of strings returned by the LDAP search. The prefix will be removed from the original strings and the resulting strings will be treated as local role names. Empty by default.

## Data Backup

While `replication` 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 `you can't just drop tables with a MergeTree-like engine containing more than 50 Gb of data`. However, these safeguards do not cover all possible cases and can be circumvented.

In order to effectively mitigate possible human errors, you should carefully prepare a strategy for backing up and restoring your data **in advance**.

Each company has different resources available and business requirements, so there's no universal solution for ClickHouse backups and restores that will fit every situation. What works for one gigabyte of data likely won't work for tens of petabytes. There are a variety of possible approaches with their own pros and cons, which will be discussed below. It is a good idea to use several approaches instead of just one in order to compensate for their various shortcomings.

### Note

Keep in mind that if you backed something up and never tried to restore it, chances are that restore will not work properly when you actually need it (or at least it will take longer than business can tolerate). So whatever backup approach you choose, make sure to automate the restore process as well, and practice it on a spare ClickHouse cluster regularly.

# Duplicating Source Data Somewhere Else

Often data that is ingested into ClickHouse is delivered through some sort of persistent queue, such as [Apache Kafka](#). In this case it is possible to configure an additional set of subscribers that will read the same data stream while it is being written to ClickHouse and store it in cold storage somewhere. Most companies already have some default recommended cold storage, which could be an object store or a distributed filesystem like [HDFS](#).

## Filesystem Snapshots

Some local filesystems provide snapshot functionality (for example, [ZFS](#)), but they might not be the best choice for serving live queries. A possible solution is to create additional replicas with this kind of filesystem and exclude them from the [Distributed](#) tables that are used for `SELECT` queries. Snapshots on such replicas will be out of reach of any queries that modify data. As a bonus, these replicas might have special hardware configurations with more disks attached per server, which would be cost-effective.

## `clickhouse-copier`

[clickhouse-copier](#) is a versatile tool that was initially created to re-shard petabyte-sized tables. It can also be used for backup and restore purposes because it reliably copies data between ClickHouse tables and clusters.

For smaller volumes of data, a simple `INSERT INTO ... SELECT ...` to remote tables might work as well.

## Manipulations with Parts

ClickHouse allows using the `ALTER TABLE ... FREEZE PARTITION ...` query to create a local copy of table partitions. This is implemented using hardlinks to the `/var/lib/clickhouse/shadow/` folder, so it usually does not consume extra disk space for old data. The created copies of files are not handled by ClickHouse server, so you can just leave them there: you will have a simple backup that does not require any additional external system, but it will still be prone to hardware issues. For this reason, it's better to remotely copy them to another location and then remove the local copies. Distributed filesystems and object stores are still a good options for this, but normal attached file servers with a large enough capacity might work as well (in this case the transfer will occur via the network filesystem or maybe [rsync](#)).

Data can be restored from backup using the `ALTER TABLE ... ATTACH PARTITION ...`.

For more information about queries related to partition manipulations, see the [ALTER documentation](#).

A third-party tool is available to automate this approach: [clickhouse-backup](#).

## Configuration Files

ClickHouse supports multi-file configuration management. The main server configuration file is `/etc/clickhouse-server/config.xml` or `/etc/clickhouse-server/config.yaml`. Other files must be in the `/etc/clickhouse-server/config.d` directory. Note, that any configuration file can be written either in XML or YAML, but mixing formats in one file is not supported. For example, you can have main configs as `config.xml` and `users.xml` and write additional files in `config.d` and `users.d` directories in `.yaml`.

All XML files should have the same root element, usually `<clickhouse>`. As for YAML, `clickhouse:` should not be present, the parser will insert it automatically.

## Override

Some settings specified in the main configuration file can be overridden in other configuration files:

- The `replace` or `remove` attributes can be specified for the elements of these configuration files.

- If neither is specified, it combines the contents of elements recursively, replacing values of duplicate children.
- If `replace` is specified, it replaces the entire element with the specified one.
- If `remove` is specified, it deletes the element.

You can also declare attributes as coming from environment variables by using `from_env="VARIABLE_NAME"`:

```
<clickhouse>
  <macros>
    <replica from_env="REPLICA" />
    <layer from_env="LAYER" />
    <shard from_env="SHARD" />
  </macros>
</clickhouse>
```

## Substitution

The config can also define “substitutions”. If an element has the `incl` attribute, the corresponding substitution from the file will be used as the value. By default, the path to the file with substitutions is `/etc/metrika.xml`. This can be changed in the `include_from` element in the server config. The substitution values are specified in `/clickhouse/substitution_name` elements in this file. If a substitution specified in `incl` does not exist, it is recorded in the log. To prevent ClickHouse from logging missing substitutions, specify the `optional="true"` attribute (for example, settings for `macros`).

If you want to replace an entire element with a substitution use `include` as element name.

XML substitution example:

```
<clickhouse>
  <!-- Appends XML subtree found at `'/profiles-in-zookeeper` ZK path to `<profiles>` element. -->
  <profiles from_zk="/profiles-in-zookeeper" />

  <users>
    <!-- Replaces `include` element with the subtree found at `'/users-in-zookeeper` ZK path. -->
    <include from_zk="/users-in-zookeeper" />
    <include from_zk="/other-users-in-zookeeper" />
  </users>
</clickhouse>
```

Substitutions can also be performed from ZooKeeper. To do this, specify the attribute `from_zk = "/path/to/node"`. The element value is replaced with the contents of the node at `/path/to/node` in ZooKeeper. You can also put an entire XML subtree on the ZooKeeper node and it will be fully inserted into the source element.

## User Settings

The `config.xml` file can specify a separate config with user settings, profiles, and quotas. The relative path to this config is set in the `users_config` element. By default, it is `users.xml`. If `users_config` is omitted, the user settings, profiles, and quotas are specified directly in `config.xml`.

Users configuration can be splitted into separate files similar to `config.xml` and `config.d/`. Directory name is defined as `users_config` setting without `.xml` postfix concatenated with `.d`. Directory `users.d` is used by default, as `users_config` defaults to `users.xml`.

Note that configuration files are first merged taking into account `Override` settings and includes are processed after that.

## XML example

For example, you can have separate config file for each user like this:

```
$ cat /etc/clickhouse-server/users.d/alice.xml
```

```
<clickhouse>
  <users>
    <alice>
      <profile>analytics</profile>
      <networks>
        <ip>::/0</ip>
      </networks>
      <password_sha256_hex>...</password_sha256_hex>
      <quota>analytics</quota>
    </alice>
  </users>
</clickhouse>
```

## YAML examples

Here you can see default config written in YAML: [config.yaml.example](#).

There are some differences between YAML and XML formats in terms of ClickHouse configurations. Here are some tips for writing a configuration in YAML format.

You should use a Scalar node to write a key-value pair:

```
key: value
```

To create a node, containing other nodes you should use a Map:

```
map_key:
  key1: val1
  key2: val2
  key3: val3
```

To create a list of values or nodes assigned to one tag you should use a Sequence:

```
seq_key:
  - val1
  - val2
  - key1: val3
  - map:
    key2: val4
    key3: val5
```

If you want to write an attribute for a Sequence or Map node, you should use a @ prefix before the attribute key. Note, that @ is reserved by YAML standard, so you should also wrap it into double quotes:

```
map:
  "@attr1": value1
  "@attr2": value2
  key: 123
```

From that Map we will get these XML nodes:

```
<map attr1="value1" attr2="value2">
  <key>123</key>
</map>
```

You can also set attributes for Sequence:

```
seq:  
  - "@attr1": value1  
  - "@attr2": value2  
  - 123  
  - abc
```

So, we can get YAML config equal to this XML one:

```
<seq attr1="value1" attr2="value2">123</seq>  
<seq attr1="value1" attr2="value2">abc</seq>
```

## Implementation Details

For each config file, the server also generates `file-preprocessed.xml` files when starting. These files contain all the completed substitutions and overrides, and they are intended for informational use. If ZooKeeper substitutions were used in the config files but ZooKeeper is not available on the server start, the server loads the configuration from the preprocessed file.

The server tracks changes in config files, as well as files and ZooKeeper nodes that were used when performing substitutions and overrides, and reloads the settings for users and clusters on the fly. This means that you can modify the cluster, users, and their settings without restarting the server.

## Quotas

Quotas allow you to limit resource usage over a period of time or track the use of resources.

Quotas are set up in the user config, which is usually '`users.xml`'.

The system also has a feature for limiting the complexity of a single query. See the section [Restrictions on query complexity](#).

In contrast to query complexity restrictions, quotas:

- Place restrictions on a set of queries that can be run over a period of time, instead of limiting a single query.
- Account for resources spent on all remote servers for distributed query processing.

Let's look at the section of the '`users.xml`' file that defines quotas.

```
<!-- 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>  
      <query_selects>0</query_selects>  
      <query_inserts>0</query_inserts>  
      <errors>0</errors>  
      <result_rows>0</result_rows>  
      <read_rows>0</read_rows>  
      <execution_time>0</execution_time>  
    </interval>  
  </default>
```

By default, the quota tracks resource consumption for each hour, without limiting usage. The resource consumption calculated for each interval is output to the server log after each request.

```
<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>
    <query_selects>100</query_selects>
    <query_inserts>100</query_inserts>
    <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>
    <query_selects>10000</query_selects>
    <query_inserts>10000</query_inserts>
    <errors>1000</errors>
    <result_rows>5000000000</result_rows>
    <read_rows>500000000000</read_rows>
    <execution_time>7200</execution_time>
  </interval>
</statbox>
```

For the ‘statbox’ quota, restrictions are set for every hour and for every 24 hours (86,400 seconds). The time interval is counted, starting from an implementation-defined fixed moment in time. In other words, the 24-hour interval does not necessarily begin at midnight.

When the interval ends, all collected values are cleared. For the next hour, the quota calculation starts over.

Here are the amounts that can be restricted:

`queries` – The total number of requests.

`query_selects` – The total number of select requests.

`query_inserts` – The total number of insert requests.

`errors` – The number of queries that threw an exception.

`result_rows` – The total number of rows given as a 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).

If the limit is exceeded for at least one time interval, an exception is thrown with a text about which restriction was exceeded, for which interval, and when the new interval begins (when queries can be sent again).

Quotas can use the “quota key” feature 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 />

```

The quota is assigned to users in the ‘users’ section of the config. See the section “Access rights”.

For distributed query processing, the accumulated amounts are stored on the requestor server. So if the user goes to another server, the quota there will “start over”.

When the server is restarted, quotas are reset.

## Optimizing Performance

- [Sampling query profiler](#)

## Sampling Query Profiler

ClickHouse runs sampling profiler that allows analyzing query execution. Using profiler you can find source code routines that used the most frequently during query execution. You can trace CPU time and wall-clock time spent including idle time.

To use profiler:

- Setup the [trace\\_log](#) section of the server configuration.

This section configures the [trace\\_log](#) system table containing the results of the profiler functioning. It is configured by default. Remember that data in this table is valid only for a running server. After the server restart, ClickHouse does not clean up the table and all the stored virtual memory address may become invalid.

- Setup the [query\\_profiler\\_cpu\\_time\\_period\\_ns](#) or [query\\_profiler\\_real\\_time\\_period\\_ns](#) settings. Both settings can be used simultaneously.

These settings allow you to configure profiler timers. As these are the session settings, you can get different sampling frequency for the whole server, individual users or user profiles, for your interactive session, and for each individual query.

The default sampling frequency is one sample per second and both CPU and real timers are enabled. This frequency allows collecting enough information about ClickHouse cluster. At the same time, working with this frequency, profiler does not affect ClickHouse server’s performance. If you need to profile each individual query try to use higher sampling frequency.

To analyze the [trace\\_log](#) system table:

- Install the [clickhouse-common-static-dbg](#) package. See [Install from DEB Packages](#).
- Allow introspection functions by the [allow\\_introspection\\_functions](#) setting.

For security reasons, introspection functions are disabled by default.

- Use the `addressToLine`, `addressToSymbol` and `demangle` [introspection functions](#) to get function names and their positions in ClickHouse code. To get a profile for some query, you need to aggregate data from the `trace_log` table. You can aggregate data by individual functions or by the whole stack traces.

If you need to visualize `trace_log` info, try [flamegraph](#) and [speedscope](#).

## Example

In this example we:

- Filtering `trace_log` data by a query identifier and the current date.
- Aggregating by stack trace.
- Using introspection functions, we will get a report of:
  - Names of symbols and corresponding source code functions.
  - Source code locations of these functions.

```
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, 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
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<DB::ReadBuffer*> (std::vector<DB::IDataType::Substream,
std::allocator<DB::IDataType::Substream> > const&),
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)::_
{lambda(bool)#1}::operator()(bool) const::{lambda(std::vector<DB::IDataType::Substream,
std::allocator<DB::IDataType::Substream> > const&#1>::_M_invoke(std::any 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::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
DR::MergeTreeRangeReader::continueReadingChain(DR::MergeTreeRangeReader::ReadResult&)
```

```

DB::MergeTreeRangeReader::cont under reading chain(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&)

```

```
__atomic_sub<__int64>(&__atomic_get, &__atomic_get)
/usr/local/include/c++/9.1.0/bits/atomic_base.h:551
ThreadPoolImpl<ThreadFromGlobalPool>::worker(std::list<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<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,
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>::*&&)
```

```
DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::operator() const
    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 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/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>::executImpl(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++-
```

```

/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/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, 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 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)
 /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<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

```

```

__III_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::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >, double, std::less<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > >, std::allocator<std::pair<std::__cxx11::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::__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
```

```
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<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

```

# System Tables

## Introduction

System tables provide information about:

- Server states, processes, and environment.
- Server's internal processes.

System tables:

- Located in the `system` database.
- Available only for reading data.
- Can't be dropped or altered, but can be detached.

Most of system tables store their data in RAM. A ClickHouse server creates such system tables at the start.

Unlike other system tables, the system log tables `metric_log`, `query_log`, `query_thread_log`, `trace_log`, `part_log`, `crash_log` and `text_log` are served by `MergeTree` table engine and store their data in a filesystem by default. If you remove a table from a filesystem, the ClickHouse server creates the empty one again at the time of the next data writing. If system table schema changed in a new release, then ClickHouse renames the current table and creates a new one.

System log tables can be customized by creating a config file with the same name as the table under `/etc/clickhouse-server/config.d/`, or setting corresponding elements in `/etc/clickhouse-server/config.xml`. Elements can be customized are:

- `database`: database the system log table belongs to. This option is deprecated now. All system log tables are under database `system`.
- `table`: table to insert data.
- `partition_by`: specify `PARTITION BY` expression.
- `ttl`: specify table `TTL` expression.
- `flush_interval_milliseconds`: interval of flushing data to disk.
- `engine`: provide full engine expression (starting with `ENGINE =` ) with parameters. This option is contradict with `partition_by` and `ttl`. If set together, the server would raise an exception and exit.

An example:

```
<clickhouse>
  <query_log>
    <database>system</database>
    <table>query_log</table>
    <partition_by>toYYYYMM(event_date)</partition_by>
    <ttl>event_date + INTERVAL 30 DAY DELETE</ttl>
    <!--
      <engine>ENGINE = MergeTree PARTITION BY toYYYYMM(event_date) ORDER BY (event_date, event_time)
      SETTINGS index_granularity = 1024</engine>
    -->
    <flush_interval_milliseconds>7500</flush_interval_milliseconds>
  </query_log>
</clickhouse>
```

By default, table growth is unlimited. To control a size of a table, you can use `TTL` settings for removing outdated log records. Also you can use the partitioning feature of `MergeTree`-engine tables.

## Sources of System Metrics

For collecting system metrics ClickHouse server uses:

- `CAP_NET_ADMIN` capability.
- `procfs` (only in Linux).

### procfs

If ClickHouse server does not have `CAP_NET_ADMIN` capability, it tries to fall back to `ProcfsMetricsProvider`. `ProcfsMetricsProvider` allows collecting per-query system metrics (for CPU and I/O).

If `procfs` is supported and enabled on the system, ClickHouse server collects these metrics:

- `OSSCPUVirtualTimeMicroseconds`
- `OSSCPUWaitMicroseconds`
- `OSIOWaitMicroseconds`

- OSReadChars
- OSWriteChars
- OSReadBytes
- OSWriteBytes

## system.asynchronous\_metric\_log

Contains the historical values for `system.asynchronous_metrics`, which are saved once per minute. Enabled by default.

Columns:

- `event_date` (**Date**) — Event date.
- `event_time` (**DateTime**) — Event time.
- `event_time_microseconds` (**DateTime64**) — Event time with microseconds resolution.
- `name` (**String**) — Metric name.
- `value` (**Float64**) — Metric value.

### Example

```
SELECT * FROM system.asynchronous_metric_log LIMIT 10
```

event_date	event_time	event_time_microseconds	name	value
2020-09-05	2020-09-05 15:56:30	2020-09-05 15:56:30.025227	CPUFrequencyMHz_0	2120.9
2020-09-05	2020-09-05 15:56:30	2020-09-05 15:56:30.025227	jemalloc.arenas.all.pmuzzy	743
2020-09-05	2020-09-05 15:56:30	2020-09-05 15:56:30.025227	jemalloc.arenas.all.pdirty	26288
2020-09-05	2020-09-05 15:56:30	2020-09-05 15:56:30.025227	jemalloc.background_thread.run_intervals	0
2020-09-05	2020-09-05 15:56:30	2020-09-05 15:56:30.025227	jemalloc.background_thread.num_runs	0
2020-09-05	2020-09-05 15:56:30	2020-09-05 15:56:30.025227	jemalloc.retained	60694528
2020-09-05	2020-09-05 15:56:30	2020-09-05 15:56:30.025227	jemalloc.mapped	303161344
2020-09-05	2020-09-05 15:56:30	2020-09-05 15:56:30.025227	jemalloc.resident	260931584
2020-09-05	2020-09-05 15:56:30	2020-09-05 15:56:30.025227	jemalloc.metadata	12079488
2020-09-05	2020-09-05 15:56:30	2020-09-05 15:56:30.025227	jemalloc.allocated	133756128

### See Also

- [system.asynchronous\\_metrics](#) — Contains metrics, calculated periodically in the background.
- [system.metric\\_log](#) — Contains history of metrics values from tables `system.metrics` and `system.events`, periodically flushed to disk.

## system.asynchronous\_metrics

Contains metrics that are calculated periodically in the background. For example, the amount of RAM in use.

Columns:

- `metric` ([String](#)) — Metric name.
- `value` ([Float64](#)) — Metric value.

## Example

```
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

## See Also

- [Monitoring](#) — Base concepts of ClickHouse monitoring.
- [system.metrics](#) — Contains instantly calculated metrics.
- [system.events](#) — Contains a number of events that have occurred.
- [system.metric\\_log](#) — Contains a history of metrics values from tables `system.metrics` and `system.events`.

# system.clusters

Contains information about clusters available in the config file and the servers in them.

Columns:

- `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.
- `is_local` ([UInt8](#)) — Flag that indicates whether the host is local.
- `user` ([String](#)) — The name of the user for connecting to the server.
- `default_database` ([String](#)) — The default database name.
- `errors_count` ([UInt32](#)) — The number of times this host failed to reach replica.

- `slowdowns_count` (`UInt32`) — The number of slowdowns that led to changing replica when establishing a connection with hedged requests.
- `estimated_recovery_time` (`UInt32`) — Seconds remaining until the replica error count is zeroed and it is considered to be back to normal.

## Example

Query:

```
SELECT * FROM system.clusters LIMIT 2 FORMAT Vertical;
```

Result:

Row 1:

```
cluster:          test_cluster_two_shards
shard_num:        1
shard_weight:     1
replica_num:      1
host_name:        127.0.0.1
host_address:     127.0.0.1
port:             9000
is_local:         1
user:             default
default_database:
errors_count:     0
slowdowns_count:  0
estimated_recovery_time: 0
```

Row 2:

```
cluster:          test_cluster_two_shards
shard_num:        2
shard_weight:     1
replica_num:      1
host_name:        127.0.0.2
host_address:     127.0.0.2
port:             9000
is_local:         0
user:             default
default_database:
errors_count:     0
slowdowns_count:  0
estimated_recovery_time: 0
```

## See Also

- [Table engine Distributed](#)
- [distributed\\_replica\\_error\\_cap setting](#)
- [distributed\\_replica\\_error\\_half\\_life setting](#)

## system.columns

Contains information about columns in all the tables.

You can use this table to get information similar to the [DESCRIBE TABLE](#) query, but for multiple tables at once.

Columns from [temporary tables](#) are visible in the `system.columns` only in those session where they have been created. They are shown with the empty `database` field.

## Columns:

- `database` ([String](#)) — Database name.
- `table` ([String](#)) — Table name.
- `name` ([String](#)) — Column name.
- `type` ([String](#)) — Column type.
- `position` ([UInt64](#)) — Ordinal position of a column in a table starting with 1.
- `default_kind` ([String](#)) — Expression type (DEFAULT, MATERIALIZED, ALIAS) for the default value, or an empty string if it is not defined.
- `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.
- `compression_codec` ([String](#)) — Compression codec name.

## Example

```
SELECT * FROM system.columns LIMIT 2 FORMAT Vertical;
```

Row 1:

```
database:      system
table:        aggregate_function_combinators
name:         name
type:          String
default_kind:
default_expression:
data_compressed_bytes: 0
data_uncompressed_bytes: 0
marks_bytes:    0
comment:
is_in_partition_key: 0
is_in_sorting_key:   0
is_in_primary_key:   0
is_in_sampling_key:  0
compression_codec:
```

Row 2:

```
database:      system
table:        aggregate_function_combinators
name:         is_internal
type:          UInt8
default_kind:
default_expression:
data_compressed_bytes: 0
data_uncompressed_bytes: 0
marks_bytes:    0
comment:
is_in_partition_key: 0
is_in_sorting_key:   0
is_in_primary_key:   0
is_in_sampling_key:  0
compression_codec:
```

The `system.columns` table contains the following columns (the column type is shown in brackets):

- `database` (String) — Database name.
- `table` (String) — Table name.
- `name` (String) — Column name.
- `type` (String) — Column type.
- `default_kind` (String) — Expression type (DEFAULT, MATERIALIZED, ALIAS) for the default value, or an empty string if it is not defined.
- `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.

# system.contributors

Contains information about contributors. The order is random at query execution time.

Columns:

- `name` (String) — Contributor (author) name from git log.

## Example

```
SELECT * FROM system.contributors LIMIT 10
```

name
Olga Khvostikova
Max Vetrov
LiuYangkuan
svladykin
zamulla
Šimon Podlipský
BayoNet
Ilya Khomutov
Amy Krishnevsky
Loud_Scream

To find out yourself in the table, use a query:

```
SELECT * FROM system.contributors WHERE name = 'Olga Khvostikova'
```

name
Olga Khvostikova

# system.crash\_log

Contains information about stack traces for fatal errors. The table does not exist in the database by default, it is created only when fatal errors occur.

Columns:

- `event_date` (Datetime) — Date of the event.
- `event_time` (Datetime) — Time of the event.
- `timestamp_ns` (UInt64) — Timestamp of the event with nanoseconds.
- `signal` (Int32) — Signal number.
- `thread_id` (UInt64) — Thread ID.
- `query_id` (String) — Query ID.
- `trace` (Array(UInt64)) — Stack trace at the moment of crash. Each element is a virtual memory address inside ClickHouse server process.
- `trace_full` (Array(String)) — Stack trace at the moment of crash. Each element contains a called method inside ClickHouse server process.

- `version` ([String](#)) — ClickHouse server version.
- `revision` ([UInt32](#)) — ClickHouse server revision.
- `build_id` ([String](#)) — BuildID that is generated by compiler.

## Example

Query:

```
SELECT * FROM system.crash_log ORDER BY event_time DESC LIMIT 1;
```

Result (not full):

```
Row 1:  
_____  
event_date: 2020-10-14  
event_time: 2020-10-14 15:47:40  
timestamp_ns: 1602679660271312710  
signal: 11  
thread_id: 23624  
query_id: 428aab7c-8f5c-44e9-9607-d16b44467e69  
trace: [188531193,...]  
trace_full: ['3. DB::(anonymous  
namespace)::FunctionFormatReadableTimeDelta::executImpl(std::__1::vector<DB::ColumnWithTypeName,  
std::__1::allocator<DB::ColumnWithTypeName> >&, std::__1::vector<unsigned long, std::__1::allocator<unsigned  
long> > const&, unsigned long, unsigned long) const @ 0xb3cc1f9 in  
/home/username/work/ClickHouse/build/programs/clickhouse',...]  
version: ClickHouse 20.11.1.1  
revision: 54442  
build_id:
```

## See also

- [trace\\_log](#) system table

[Original article](#)

## system.current\_roles

Contains active roles of a current user. `SET ROLE` changes the contents of this table.

Columns:

- `role_name` ([String](#)) — Role name.
- `with_admin_option` ([UInt8](#)) — Flag that shows whether `current_role` is a role with `ADMIN OPTION` privilege.
- `is_default` ([UInt8](#)) — Flag that shows whether `current_role` is a default role.

## system.data\_skipping\_indices

Contains information about existing data skipping indices in all the tables.

Columns:

- `database` ([String](#)) — Database name.
- `table` ([String](#)) — Table name.
- `name` ([String](#)) — Index name.

- `type` ([String](#)) — Index type.
- `expr` ([String](#)) — Expression for the index calculation.
- `granularity` ([UInt64](#)) — The number of granules in the block.
- `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.

## Example

```
SELECT * FROM system.data_skipping_indices LIMIT 2 FORMAT Vertical;
```

Row 1:

```
database: default
table: user_actions
name: clicks_idx
type: minmax
expr: clicks
granularity: 1
data_compressed_bytes: 58
data_uncompressed_bytes: 6
marks: 48
```

Row 2:

```
database: default
table: users
name: contacts_null_idx
type: minmax
expr: assumeNotNull(contacts_null)
granularity: 1
data_compressed_bytes: 58
data_uncompressed_bytes: 6
marks: 48
```

## system.data\_type\_families

Contains information about supported [data types](#).

Columns:

- `name` ([String](#)) — Data type name.
- `case_insensitive` ([UInt8](#)) — Property that shows whether you can use a data type name in a query in case insensitive manner or not. For example, `Date` and `date` are both valid.
- `alias_to` ([String](#)) — Data type name for which `name` is an alias.

## Example

```
SELECT * FROM system.data_type_families WHERE alias_to = 'String'
```

name	case_insensitive	alias_to
LONGBLOB	1   String	
LONGTEXT	1   String	
TINYTEXT	1   String	
TEXT	1   String	
VARCHAR	1   String	
MEDIUMBLOB	1   String	
BLOB	1   String	
TINYBLOB	1   String	
CHAR	1   String	
MEDIUMTEXT	1   String	

## See Also

- [Syntax](#) — Information about supported syntax.

## system.databases

Contains information about the databases that are available to the current user.

Columns:

- `name` ([String](#)) — Database name.
- `engine` ([String](#)) — Database engine.
- `data_path` ([String](#)) — Data path.
- `metadata_path` ([String](#)) — Metadata path.
- `uuid` ([UUID](#)) — Database UUID.

The `name` column from this system table is used for implementing the `SHOW DATABASES` query.

## Example

Create a database.

```
CREATE DATABASE test
```

Check all of the available databases to the user.

```
SELECT * FROM system.databases
```

name	engine	data_path	metadata_path	uuid
_temporary_and_external_tables	Memory	/var/lib/clickhouse/		
00000000-0000-0000-0000-000000000000				
default	Atomic	/var/lib/clickhouse/store/	/var/lib/clickhouse/store/d31/d317b4bd-3595-4386-81ee-c2334694128a/	d317b4bd-3595-4386-81ee-c2334694128a
test	Atomic	/var/lib/clickhouse/store/	/var/lib/clickhouse/store/39b/39bf0cc5-4c06-4717-87fe-c75ff3bd8ebb/	39bf0cc5-4c06-4717-87fe-c75ff3bd8ebb
system	Atomic	/var/lib/clickhouse/store/	/var/lib/clickhouse/store/1d1/1d1c869d-e465-4b1b-a51fbe033436ebf9/	1d1c869d-e465-4b1b-a51fbe033436ebf9

## system.detached\_parts

Contains information about detached parts of **MergeTree** tables. The `reason` column specifies why the part was detached.

For user-detached parts, the reason is empty. Such parts can be attached with **ALTER TABLE ATTACH PARTITION|PART** command.

For the description of other columns, see [system.parts](#).

If part name is invalid, values of some columns may be `NULL`. Such parts can be deleted with **ALTER TABLE DROP DETACHED PART**.

## system.dictionaries

Contains information about [external dictionaries](#).

Columns:

- `database` ([String](#)) — Name of the database containing the dictionary created by DDL query. Empty string for other dictionaries.
- `name` ([String](#)) — Dictionary name.
- `status` ([Enum8](#)) — 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](#) query, timeout, dictionary config has changed).
  - `FAILED_AND_RELOADING` — Could not load the dictionary as a result of an error and is loading now.
- `origin` ([String](#)) — Path to the configuration file that describes the dictionary.
- `type` ([String](#)) — Type of a dictionary allocation. [Storing Dictionaries in Memory](#).
- `key` — **Key type**: Numeric Key ([UInt64](#)) or Composite key ([String](#)) — form “(type 1, type 2, ..., type n)”.
- `attribute.names` ([Array\(String\)](#)) — Array of [attribute names](#) provided by the dictionary.
- `attribute.types` ([Array\(String\)](#)) — Corresponding array of [attribute types](#) that are provided by the dictionary.
- `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.
- `found_rate` ([Float64](#)) — The percentage of uses for which the value was found.
- `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` (`String`) — Text describing the **data source** for the dictionary.
- `lifetime_min` (`UInt64`) — Minimum **lifetime** of the dictionary in memory, after which ClickHouse tries to reload the dictionary (if `invalidate_query` is set, then only if it has changed). Set in seconds.
- `lifetime_max` (`UInt64`) — Maximum **lifetime** of the dictionary in memory, after which ClickHouse tries to reload the dictionary (if `invalidate_query` is set, then only if it has changed). Set in seconds.
- `loading_start_time` (`DateTime`) — Start time for loading the dictionary.
- `last_successful_update_time` (`DateTime`) — 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` (`String`) — Text of the error that occurs when creating or reloading the dictionary if the dictionary couldn't be created.

## Example

Configure the dictionary.

```
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())
```

Make sure that the dictionary is loaded.

```
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	

## system.disks

Contains information about disks defined in the **server configuration**.

Columns:

- `name` (`String`) — Name of a disk in the server configuration.
- `path` (`String`) — 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` parameter of disk configuration.

## Example

```
:) SELECT * FROM system.disks;
```

name	path	free_space	total_space	keep_free_space
default	/var/lib/clickhouse/	276392587264	490652508160	0

1 rows in set. Elapsed: 0.001 sec.

## system.distributed\_ddl\_queue

Contains information about [distributed ddl queries \(ON CLUSTER clause\)](#) that were executed on a cluster.

Columns:

- `entry` (`String`) — Query id.
- `host_name` (`String`) — Hostname.
- `host_address` (`String`) — IP address that the Hostname resolves to.
- `port` (`UInt16`) — Host Port.
- `status` (`Enum8`) — Status of the query.
- `cluster` (`String`) — Cluster name.
- `query` (`String`) — Query executed.
- `initiator` (`String`) — Node that executed the query.
- `query_start_time` (`DateTime`) — Query start time.
- `query_finish_time` (`DateTime`) — Query finish time.
- `query_duration_ms` (`UInt64`) — Duration of query execution (in milliseconds).
- `exception_code` (`Enum8`) — Exception code from [ZooKeeper](#).

## Example

```
SELECT *
FROM system.distributed_ddl_queue
WHERE cluster = 'test_cluster'
LIMIT 2
FORMAT Vertical

Query id: f544e72a-6641-43f1-836b-24baa1c9632a

Row 1:
_____
entry:      query-000000000000
host_name:  clickhouse01
host_address: 172.23.0.11
port:      9000
status:    Finished
cluster:   test_cluster
query:     CREATE DATABASE test_db UUID '4a82697e-c85e-4e5b-a01e-a36f2a758456' ON CLUSTER test_cluster
initiator: clickhouse01:9000
query_start_time: 2020-12-30 13:07:51
query_finish_time: 2020-12-30 13:07:51
query_duration_ms: 6
exception_code: ZOK

Row 2:
_____
entry:      query-000000000000
host_name:  clickhouse02
host_address: 172.23.0.12
port:      9000
status:    Finished
cluster:   test_cluster
query:     CREATE DATABASE test_db UUID '4a82697e-c85e-4e5b-a01e-a36f2a758456' ON CLUSTER test_cluster
initiator: clickhouse01:9000
query_start_time: 2020-12-30 13:07:51
query_finish_time: 2020-12-30 13:07:51
query_duration_ms: 6
exception_code: ZOK
```

2 rows in set. Elapsed: 0.025 sec.

## system.distribution\_queue

Contains information about local files that are in the queue to be sent to the shards. These local files contain new parts that are created by inserting new data into the Distributed table in asynchronous mode.

Columns:

- `database` ([String](#)) — Name of the database.
- `table` ([String](#)) — Name of the table.
- `data_path` ([String](#)) — Path to the folder with local files.
- `is_blocked` ([UInt8](#)) — Flag indicates whether sending local files to the server is blocked.
- `error_count` ([UInt64](#)) — Number of errors.
- `data_files` ([UInt64](#)) — Number of local files in a folder.
- `data_compressed_bytes` ([UInt64](#)) — Size of compressed data in local files, in bytes.
- `broken_data_files` ([UInt64](#)) — Number of files that has been marked as broken (due to an error).
- `broken_data_compressed_bytes` ([UInt64](#)) — Size of compressed data in broken files, in bytes.
- `last_exception` ([String](#)) — Text message about the last error that occurred (if any).

## Example

```
SELECT * FROM system.distribution_queue LIMIT 1 FORMAT Vertical;
```

Row 1:

```
database:      default
table:        dist
data_path:     ./store/268/268bc070-3aad-4b1a-9cf2-4987580161af/default@127%2E0%2E0%2E2:9000/
is_blocked:    1
error_count:   0
data_files:    1
data_compressed_bytes: 499
last_exception:
```

## See Also

- [Distributed table engine](#)

## system.enabled\_roles

Contains all active roles at the moment, including current role of the current user and granted roles for current role.

Columns:

- `role_name` ([String](#)) — Role name.
- `with_admin_option` ([UInt8](#)) — Flag that shows whether `enabled_role` is a role with `ADMIN OPTION` privilege.
- `is_current` ([UInt8](#)) — Flag that shows whether `enabled_role` is a current role of a current user.
- `is_default` ([UInt8](#)) — Flag that shows whether `enabled_role` is a default role.

## system.errors

Contains error codes with the number of times they have been triggered.

Columns:

- `name` ([String](#)) — name of the error (`errorCodeToName`).
- `code` ([Int32](#)) — code number of the error.
- `value` ([UInt64](#)) — the number of times this error has been happened.
- `last_error_time` ([DateTime](#)) — time when the last error happened.
- `last_error_message` ([String](#)) — message for the last error.
- `last_error_trace` ([Array\(UInt64\)](#)) — A [stack trace](#) which represents a list of physical addresses where the called methods are stored.
- `remote` ([UInt8](#)) — remote exception (i.e. received during one of the distributed query).

## Example

```
SELECT name, code, value
FROM system.errors
WHERE value > 0
ORDER BY code ASC
LIMIT 1
```

name	code	value
CANNOT_OPEN_FILE	76	1

```
WITH arrayMap(x -> demangle(addressToSymbol(x)), last_error_trace) AS all
SELECT name, arrayStringConcat(all, '\n') AS res
FROM system.errors
LIMIT 1
SETTINGS allow_introspection_functions=1\G
```

## system.events

Contains information about the number of events that have occurred in the system. For example, in the table, you can find how many `SELECT` queries were processed since the ClickHouse server started.

Columns:

- `event` (`String`) — Event name.
- `value` (`UInt64`) — Number of events occurred.
- `description` (`String`) — Event description.

### Example

```
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.

### See Also

- [system.asynchronous\\_metrics](#) — Contains periodically calculated metrics.
- [system.metrics](#) — Contains instantly calculated metrics.
- [system.metric\\_log](#) — Contains a history of metrics values from tables `system.metrics` и `system.events`.

- [Monitoring](#) — Base concepts of ClickHouse monitoring.

## system.functions

Contains information about normal and aggregate functions.

Columns:

- `name(String)` — The name of the function.
- `is_aggregate(UInt8)` — Whether the function is aggregate.

### Example

```
SELECT * FROM system.functions LIMIT 10;
```

name	is_aggregate	case_insensitive	alias_to
sumburConsistentHash	0	0	
yandexConsistentHash	0	0	
demangle	0	0	
addressToLine	0	0	
JSONExtractRaw	0	0	
JSONExtractKeysAndValues	0	0	
JSONExtract	0	0	
JSONExtractString	0	0	
JSONExtractFloat	0	0	
JSONExtractInt	0	0	

10 rows in set. Elapsed: 0.002 sec.

## system.grants

Privileges granted to ClickHouse user accounts.

Columns:

- `user_name (Nullable(String))` — User name.
- `role_name (Nullable(String))` — Role assigned to user account.
- `access_type (Enum8)` — Access parameters for ClickHouse user account.
- `database (Nullable(String))` — Name of a database.
- `table (Nullable(String))` — Name of a table.
- `column (Nullable(String))` — Name of a column to which access is granted.
- `is_partial_revoke (UInt8)` — Logical value. It shows whether some privileges have been revoked. Possible values:
  - `0` — The row describes a partial revoke.
  - `1` — The row describes a grant.
- `grant_option (UInt8)` — Permission is granted WITH GRANT OPTION, see [GRANT](#).

## system.graphite\_retentions

Contains information about parameters `graphite_rollup` which are used in tables with `*GraphiteMergeTree` engines.

Columns:

- `config_name` (String) - `graphite_rollup` parameter name.
- `regexp` (String) - A pattern for the metric name.
- `function` (String) - The name of the aggregating function.
- `age` (UInt64) - The minimum age of the data in seconds.
- `precision` (UInt64) - How precisely to define the age of the data in seconds.
- `priority` (UInt16) - Pattern priority.
- `is_default` (UInt8) - Whether the pattern is the default.
- `Tables.database` (Array(String)) - Array of names of database tables that use the `config_name` parameter.
- `Tables.table` (Array(String)) - Array of table names that use the `config_name` parameter.

## system.licenses

Contains licenses of third-party libraries that are located in the `contrib` directory of ClickHouse sources.

Columns:

- `library_name` (String) — Name of the library, which is license connected with.
- `license_type` (String) — License type — e.g. Apache, MIT.
- `license_path` (String) — Path to the file with the license text.
- `license_text` (String) — License text.

### Example

```
SELECT library_name, license_type, license_path FROM system.licenses LIMIT 15
```

library_name	license_type	license_path
FastMemcpy	MIT	/contrib/FastMemcpy/LICENSE
arrow	Apache	/contrib/arrow/LICENSE.txt
avro	Apache	/contrib/avro/LICENSE.txt
aws-c-common	Apache	/contrib/aws-c-common/LICENSE
aws-c-event-stream	Apache	/contrib/aws-c-event-stream/LICENSE
aws-checksums	Apache	/contrib/aws-checksums/LICENSE
aws	Apache	/contrib/aws/LICENSE.txt
base64	BSD 2-clause	/contrib/base64/LICENSE
boost	Boost	/contrib/boost/LICENSE_1_0.txt
brotli	MIT	/contrib/brotli/LICENSE
capnproto	MIT	/contrib/capnproto/LICENSE
cassandra	Apache	/contrib/cassandra/LICENSE.txt
cctz	Apache	/contrib/cctz/LICENSE.txt
cityhash102	MIT	/contrib/cityhash102/COPYING
cppkafka	BSD 2-clause	/contrib/cppkafka/LICENSE

## system.merge\_tree\_settings

Contains information about settings for MergeTree tables.

Columns:

- `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.

## Example

```
:) SELECT * FROM system.merge_tree_settings LIMIT 4 FORMAT Vertical;
```

Row 1:

```
name:      index_granularity
value:    8192
changed:   0
description: How many rows correspond to one primary key value.
type:      SettingUInt64
```

Row 2:

```
name:      min_bytes_for_wide_part
value:    0
changed:   0
description: Minimal uncompressed size in bytes to create part in wide format instead of compact
type:      SettingUInt64
```

Row 3:

```
name:      min_rows_for_wide_part
value:    0
changed:   0
description: Minimal number of rows to create part in wide format instead of compact
type:      SettingUInt64
```

Row 4:

```
name:      merge_max_block_size
value:    8192
changed:   0
description: How many rows in blocks should be formed for merge operations.
type:      SettingUInt64
```

4 rows in set. Elapsed: 0.001 sec.

## system.merges

Contains information about merges and part mutations currently in process for tables in the MergeTree family.

Columns:

- `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) — 1 if this process is a part mutation.
- `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.
- `memory_usage` (UInt64) — Memory consumption of the merge process.
- `thread_id` (UInt64) — Thread ID of the merge process.
- `merge_type` — The type of current merge. Empty if it's an mutation.
- `merge_algorithm` — The algorithm used in current merge. Empty if it's an mutation.

## system.metric\_log

Contains history of metrics values from tables `system.metrics` and `system.events`, periodically flushed to disk.

Columns:

- `event_date` (**Date**) — Event date.
- `event_time` (**DateTime**) — Event time.
- `event_time_microseconds` (**DateTime64**) — Event time with microseconds resolution.

### Example

```
SELECT * FROM system.metric_log LIMIT 1 FORMAT Vertical;
```

Row 1:

event_date:	2020-09-05
event_time:	2020-09-05 16:22:33
event_time_microseconds:	2020-09-05 16:22:33.196807
milliseconds:	196
ProfileEvent_Query:	0
ProfileEvent_SelectQuery:	0
ProfileEvent_InsertQuery:	0
ProfileEvent_FailedQuery:	0
ProfileEvent_FailedSelectQuery:	0
...	
...	
CurrentMetric_Revision:	54439
CurrentMetric_VersionInteger:	20009001
CurrentMetric_RWLockWaitingReaders:	0
CurrentMetric_RWLockWaitingWriters:	0
CurrentMetric_RWLockActiveReaders:	0
CurrentMetric_RWLockActiveWriters:	0
CurrentMetric_GlobalThread:	74
CurrentMetric_GlobalThreadActive:	26
CurrentMetric_LocalThread:	0
CurrentMetric_LocalThreadActive:	0
CurrentMetric_DistributedFilesToInsert:	0

## See also

- [metric\\_log setting](#) — Enabling and disabling the setting.
- [system.asynchronous\\_metrics](#) — Contains periodically calculated metrics.
- [system.events](#) — Contains a number of events that occurred.
- [system.metrics](#) — Contains instantly calculated metrics.
- [Monitoring](#) — Base concepts of ClickHouse monitoring.

## system.metrics

Contains metrics which can be calculated instantly, or have a current value. For example, the number of simultaneously processed queries or the current replica delay. This table is always up to date.

Columns:

- `metric` ([String](#)) — Metric name.
- `value` ([Int64](#)) — Metric value.
- `description` ([String](#)) — Metric description.

The list of supported metrics you can find in the [src/Common/CurrentMetrics.cpp](#) source file of ClickHouse.

## Example

```
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.

## See Also

- [system.asynchronous\\_metrics](#) — Contains periodically calculated metrics.
- [system.events](#) — Contains a number of events that occurred.
- [system.metric\\_log](#) — Contains a history of metrics values from tables `system.metrics` и `system.events`.
- [Monitoring](#) — Base concepts of ClickHouse monitoring.

## system.mutations

The table contains information about [mutations](#) of [MergeTree](#) tables and their progress. Each mutation command is represented by a single row.

Columns:

- `database` ([String](#)) — The name of the database to which the mutation was applied.
- `table` ([String](#)) — The name of the table to which the mutation was applied.
- `mutation_id` ([String](#)) — The ID of the mutation. For replicated tables these IDs correspond to znode names in the `<table_path_in_zookeeper>/mutations/` directory in ZooKeeper. For non-replicated tables the IDs correspond to file names in the data directory of the table.
- `command` ([String](#)) — The mutation command string (the part of the query after `ALTER TABLE [db.]table`).
- `create_time` ([Datetime](#)) — Date and time when the mutation command was submitted for execution.
- `block_numbers.partition_id` ([Array\(String\)](#)) — For mutations of replicated tables, the array contains the partitions' IDs (one record for each partition). For mutations of non-replicated tables the array is empty.

- `block_numbers.number` ([Array\(Int64\)](#)) — For mutations of replicated tables, the array contains one record for each partition, with the block number that was acquired by the mutation. Only parts that contain blocks with numbers less than this number will be mutated in the partition.

In non-replicated tables, block numbers in all partitions form a single sequence. This means that for mutations of non-replicated tables, the column will contain one record with a single block number acquired by the mutation.

- `parts_to_do_names` ([Array\(String\)](#)) — An array of names of data parts that need to be mutated for the mutation to complete.
- `parts_to_do` ([Int64](#)) — The number of data parts that need to be mutated for the mutation to complete.
- `is_done` ([UInt8](#)) — The flag whether the mutation is done or not. Possible values:
  - `1` if the mutation is completed,
  - `0` if the mutation is still in process.

## Note

Even if `parts_to_do = 0` it is possible that a mutation of a replicated table is not completed yet because of a long-running `INSERT` query, that will create a new data part needed to be mutated.

If there were problems with mutating some data parts, the following columns contain additional information:

- `latest_failed_part` ([String](#)) — The name of the most recent part that could not be mutated.
- `latest_fail_time` ([Datetime](#)) — The date and time of the most recent part mutation failure.
- `latest_fail_reason` ([String](#)) — The exception message that caused the most recent part mutation failure.

## See Also

- [Mutations](#)
- [MergeTree](#) table engine
- [ReplicatedMergeTree](#) family

# system.numbers

This table contains a single `UInt64` column named `number` that contains almost all the natural numbers starting from zero.

You can use this table for tests, or if you need to do a brute force search.

Reads from this table are not parallelized.

## Example

```
:) SELECT * FROM system.numbers LIMIT 10;
```

number
0
1
2
3
4
5
6
7
8
9

10 rows in set. Elapsed: 0.001 sec.

## system.numbers\_mt

The same as [system.numbers](#) but reads are parallelized. The numbers can be returned in any order.

Used for tests.

### Example

```
:) SELECT * FROM system.numbers_mt LIMIT 10;
```

number
0
1
2
3
4
5
6
7
8
9

10 rows in set. Elapsed: 0.001 sec.

## system.one

This table contains a single row with a single `UInt8` column containing the value 0.

This table is used if a `SELECT` query does not specify the `FROM` clause.

This is similar to the `DUAL` table found in other DBMSs.

### Example

```
:) SELECT * FROM system.one LIMIT 10;
```

dummy
0

1 rows in set. Elapsed: 0.001 sec.

# system.opentelemetry\_span\_log

Contains information about [trace spans](#) for executed queries.

Columns:

- `trace_id` ([UUID](#)) — ID of the trace for executed query.
- `span_id` ([UInt64](#)) — ID of the `trace span`.
- `parent_span_id` ([UInt64](#)) — ID of the parent trace span.
- `operation_name` ([String](#)) — The name of the operation.
- `start_time_us` ([UInt64](#)) — The start time of the `trace span` (in microseconds).
- `finish_time_us` ([UInt64](#)) — The finish time of the `trace span` (in microseconds).
- `finish_date` ([Date](#)) — The finish date of the `trace span`.
- `attribute.names` ([Array\(String\)](#)) — Attribute names depending on the `trace span`. They are filled in according to the recommendations in the [OpenTelemetry](#) standard.
- `attribute.values` ([Array\(String\)](#)) — Attribute values depending on the `trace span`. They are filled in according to the recommendations in the [OpenTelemetry](#) standard.

## Example

Query:

```
SELECT * FROM system.opentelemetry_span_log LIMIT 1 FORMAT Vertical;
```

Result:

```
Row 1:  
_____  
trace_id: cdab0847-0d62-61d5-4d38-dd65b19a1914  
span_id: 701487461015578150  
parent_span_id: 2991972114672045096  
operation_name: DB::Block DB::InterpreterSelectQuery::getSampleBlockImpl()  
start_time_us: 1612374594529090  
finish_time_us: 1612374594529108  
finish_date: 2021-02-03  
attribute.names: []  
attribute.values: []
```

## See Also

- [OpenTelemetry](#)

# system.part\_log

The `system.part_log` table is created only if the [part\\_log](#) server setting is specified.

This table contains information about events that occurred with [data parts](#) in the [MergeTree](#) family tables, such as adding or merging data.

The `system.part_log` table contains the following columns:

- `query_id` ([String](#)) — Identifier of the `INSERT` query that created this data part.

- `event_type` ([Enum8](#)) — 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.
- `event_time_microseconds` ([DateTime64](#)) — Event time with microseconds precision.
- `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` value if the partitioning is by `tuple()`.
- `path_on_disk` ([String](#)) — Absolute path to the folder with data part files.
- `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_bytes` ([UInt64](#)) — The number of bytes was read during the merge.
- `peak_memory_usage` ([Int64](#)) — The maximum difference between the amount of allocated and freed memory in context of this thread.
- `error` ([UInt16](#)) — The code number of the occurred error.
- `exception` ([String](#)) — Text message of the occurred error.

The `system.part_log` table is created after the first inserting data to the `MergeTree` table.

## Example

```
SELECT * FROM system.part_log LIMIT 1 FORMAT Vertical;
```

Row 1:

```
query_id: 983ad9c7-28d5-4ae1-844e-603116b7de31
event_type: NewPart
event_date: 2021-02-02
event_time: 2021-02-02 11:14:28
event_time_microseconds: 2021-02-02 11:14:28.861919
duration_ms: 35
database: default
table: log_mt_2
part_name: all_1_1_0
partition_id: all
path_on_disk: db/data/default/log_mt_2/all_1_1_0/
rows: 115418
size_in_bytes: 1074311
merged_from: []
bytes_uncompressed: 0
read_rows: 0
read_bytes: 0
peak_memory_usage: 0
error: 0
exception:
```

## system.parts

Contains information about parts of [MergeTree](#) tables.

Each row describes one data part.

Columns:

- `partition` ([String](#)) – The partition name. To learn what a partition is, see the description of the [ALTER](#) query.

Formats:

- `YYYYMM` for automatic partitioning by month.
- `any_string` when partitioning manually.
- `name` ([String](#)) – Name of the data part.
- `part_type` ([String](#)) — The data part storing format.

Possible Values:

- `Wide` — Each column is stored in a separate file in a filesystem.
  - `Compact` — All columns are stored in one file in a filesystem.
- Data storing format is controlled by the `min_bytes_for_wide_part` and `min_rows_for_wide_part` settings of the [MergeTree](#) table.
- `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` by the index granularity (usually 8192) (this hint does not work for adaptive granularity).
  - `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.
- `secondary_indices_compressed_bytes` (`UInt64`) – Total size of compressed data for secondary indices in the data part. All the auxiliary files (for example, files with marks) are not included.
- `secondary_indices_uncompressed_bytes` (`UInt64`) – Total size of uncompressed data for secondary indices in the data part. All the auxiliary files (for example, files with marks) are not included.
- `secondary_indices_marks_bytes` (`UInt64`) – The size of the file with marks for secondary indices.
- `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 does not 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` of compressed files.

- `hash_of_uncompressed_files` (`String`) – `sipHash128` of uncompressed files (files with marks, index file etc.).
- `uncompressed_hash_of_compressed_files` (`String`) – `sipHash128` of data in the compressed files as if they were uncompressed.
- `delete_ttl_info_min` (`DateTime`) — The minimum value of the date and time key for **TTL DELETE rule**.
- `delete_ttl_info_max` (`DateTime`) — The maximum value of the date and time key for **TTL DELETE rule**.
- `move_ttl_info.expression` (`Array(String)`) — Array of expressions. Each expression defines a **TTL MOVE rule**.

## Warning

The `move_ttl_info.expression` array is kept mostly for backward compatibility, now the simplest way to check **TTL MOVE rule** is to use the `move_ttl_info.min` and `move_ttl_info.max` fields.

- `move_ttl_info.min` (`Array(DateTime)`) — Array of date and time values. Each element describes the minimum key value for a **TTL MOVE rule**.
- `move_ttl_info.max` (`Array(DateTime)`) — Array of date and time values. Each element describes the maximum key value for a **TTL MOVE rule**.
- `bytes` (`UInt64`) – Alias for `bytes_on_disk`.
- `marks_size` (`UInt64`) – Alias for `marks_bytes`.

## Example

```
SELECT * FROM system.parts LIMIT 1 FORMAT Vertical;
```

Row 1:

```
partition:          tuple()
name:              all_1_4_1_6
part_type:         Wide
active:            1
marks:             2
rows:              6
bytes_on_disk:    310
data_compressed_bytes: 157
data_uncompressed_bytes: 91
secondary_indices_compressed_bytes: 58
secondary_indices_uncompressed_bytes: 6
secondary_indices_marks_bytes: 48
marks_bytes:       144
modification_time: 2020-06-18 13:01:49
remove_time:       1970-01-01 00:00:00
refcount:          1
min_date:          1970-01-01
max_date:          1970-01-01
min_time:          1970-01-01 00:00:00
max_time:          1970-01-01 00:00:00
partition_id:      all
min_block_number:  1
max_block_number:  4
level:             1
data_version:     6
primary_key_bytes_in_memory: 8
primary_key_bytes_in_memory_allocated: 64
is_frozen:         0
database:          default
table:             months
engine:            MergeTree
disk_name:         default
path:              /var/lib/clickhouse/data/default/months/all_1_4_1_6/
hash_of_all_files: 2d0657a16d9430824d35e327fcdb87bf
hash_of_uncompressed_files: 84950cc30ba867c77a408ae21332ba29
uncompressed_hash_of_compressed_files: 1ad78f1c6843bbfb99a2c931abe7df7d
delete_ttl_info_min: 1970-01-01 00:00:00
delete_ttl_info_max: 1970-01-01 00:00:00
move_ttl_info.expression: []
move_ttl_info.min: []
move_ttl_info.max: []
```

## See Also

- [MergeTree family](#)
- [TTL for Columns and Tables](#)

## system.parts\_columns

Contains information about parts and columns of [MergeTree](#) tables.

Each row describes one data part.

Columns:

- **partition** ([String](#)) — The partition name. To learn what a partition is, see the description of the [ALTER](#) query.

Formats:

- `YYYYMM` for automatic partitioning by month.
- `any_string` when partitioning manually.
- **name** ([String](#)) — Name of the data part.

- `part_type` (`String`) — The data part storing format.

Possible values:

- `Wide` — Each column is stored in a separate file in a filesystem.
  - `Compact` — All columns are stored in one file in a filesystem.
- Data storing format is controlled by the `min_bytes_for_wide_part` and `min_rows_for_wide_part` settings of the [MergeTree](#) table.
- `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` by the index granularity (usually 8192) (this hint does not work for adaptive granularity).
  - `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.
  - `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.
  - `database` (`String`) — Name of the database.
  - `table` (`String`) — Name of the table.

- `engine` ([String](#)) — Name of the table engine without parameters.
- `disk_name` ([String](#)) — Name of a disk that stores the data part.
- `path` ([String](#)) — Absolute path to the folder with data part files.
- `column` ([String](#)) — Name of the column.
- `type` ([String](#)) — Column type.
- `column_position` ([UInt64](#)) — Ordinal position of a column in a table starting with 1.
- `default_kind` ([String](#)) — Expression type (DEFAULT, MATERIALIZED, ALIAS) for the default value, or an empty string if it is not defined.
- `default_expression` ([String](#)) — Expression for the default value, or an empty string if it is not defined.
- `column_bytes_on_disk` ([UInt64](#)) — Total size of the column in bytes.
- `column_data_compressed_bytes` ([UInt64](#)) — Total size of compressed data in the column, in bytes.
- `column_data_uncompressed_bytes` ([UInt64](#)) — Total size of the decompressed data in the column, in bytes.
- `column_marks_bytes` ([UInt64](#)) — The size of the column with marks, in bytes.
- `bytes` ([UInt64](#)) — Alias for `bytes_on_disk`.
- `marks_size` ([UInt64](#)) — Alias for `marks_bytes`.

## Example

```
SELECT * FROM system.parts_columns LIMIT 1 FORMAT Vertical;
```

Row 1:

```
partition:          tuple()
name:              all_1_2_1
part_type:         Wide
active:            1
marks:             2
rows:              2
bytes_on_disk:    155
data_compressed_bytes: 56
data_uncompressed_bytes: 4
marks_bytes:       96
modification_time: 2020-09-23 10:13:36
remove_time:       2106-02-07 06:28:15
refcount:          1
min_date:          1970-01-01
max_date:          1970-01-01
partition_id:      all
min_block_number:  1
max_block_number:  2
level:             1
data_version:     1
primary_key_bytes_in_memory: 2
primary_key_bytes_in_memory_allocated: 64
database:          default
table:             53r93yleapyears
engine:            MergeTree
disk_name:         default
path:              /var/lib/clickhouse/data/default/53r93yleapyears/all_1_2_1/
column:            id
type:              Int8
column_position:  1
default_kind:      default
default_expression:
column_bytes_on_disk: 76
column_data_compressed_bytes: 28
column_data_uncompressed_bytes: 2
column_marks_bytes: 48
```

## See Also

- [MergeTree family](#)

# system.processes

This system table is used for implementing the SHOW PROCESSLIST query.

Columns:

- **user** (String) – The user who made the query. Keep in mind that for distributed processing, queries are sent to remote servers under the `default` user. The field contains the username for a specific query, not for a query that this query initiated.
- **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` on the query requestor server.
- **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` setting.
- `query` (String) – The query text. For `INSERT`, it does not include the data to insert.
- `query_id` (String) – Query ID, if defined.

```
:) SELECT * FROM system.processes LIMIT 10 FORMAT Vertical;
```

Row 1:

```
is_initial_query: 1
user: default
query_id: 35a360fa-3743-441d-8e1f-228c938268da
address: ::ffff:172.23.0.1
port: 47588
initial_user: default
initial_query_id: 35a360fa-3743-441d-8e1f-228c938268da
initial_address: ::ffff:172.23.0.1
initial_port: 47588
interface: 1
os_user: bharatnc
client_hostname: tower
client_name: ClickHouse
client_revision: 54437
client_version_major: 20
client_version_minor: 7
client_version_patch: 2
http_method: 0
http_user_agent:
quota_key:
elapsed: 0.000582537
is_cancelled: 0
read_rows: 0
read_bytes: 0
total_rows_approx: 0
written_rows: 0
written_bytes: 0
memory_usage: 0
peak_memory_usage: 0
query: SELECT * from system.processes LIMIT 10 FORMAT Vertical;
thread_ids: [67]
ProfileEvents:
{'Query':1,'SelectQuery':1,'ReadCompressedBytes':36,'CompressedReadBufferBlocks':1,'CompressedReadBufferBytes':10,'IOBufferAllocs':1,'IOBufferAllocBytes':89,'ContextLock':15,'RWLockAcquiredReadLocks':1}
Settings:
{'background_pool_size':'32','load_balancing':'random','allow_suspicious_low_cardinality_types':'1','distributed_aggregation_memory_efficient':'1','skip_unavailable_shards':'1','log_queries':'1','max_bytes_before_external_group_by':'20000000000','max_bytes_before_external_sort':'20000000000','allow_introspection_functions':'1'}
1 rows in set. Elapsed: 0.002 sec.
```

## system.query\_log

Contains information about executed queries, for example, start time, duration of processing, error messages.

### Note

This table does not contain the ingested data for `INSERT` queries.

You can change settings of queries logging in the `query_log` section of the server configuration.

You can disable queries logging by setting `log_queries = 0`. We do not recommend to turn off logging because information in this table is important for solving issues.

The flushing period of data is set in `flush_interval_milliseconds` parameter of the `query_log` server settings section. To force flushing, use the `SYSTEM FLUSH LOGS` query.

ClickHouse does not delete data from the table automatically. See [Introduction](#) for more details.

The `system.query_log` table registers two kinds of queries:

1. Initial queries that were run directly by the client.
2. Child queries that were initiated by other queries (for distributed query execution). For these types of queries, information about the parent queries is shown in the `initial_*` columns.

Each query creates one or two rows in the `query_log` table, depending on the status (see the `type` column) of the query:

1. If the query execution was successful, two rows with the `QueryStart` and `QueryFinish` types are created.
2. If an error occurred during query processing, two events with the `QueryStart` and `ExceptionWhileProcessing` types are created.
3. If an error occurred before launching the query, a single event with the `ExceptionBeforeStart` type is created.

You can use the `log_queries_probability` setting to reduce the number of queries, registered in the `query_log` table.

Columns:

- `type` (`Enum8`) — Type of an 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.
- `event_time_microseconds` (`DateTime`) — Query starting time with microseconds precision.
- `query_start_time` (`DateTime`) — Start time of query execution.
- `query_start_time_microseconds` (`DateTime64`) — Start time of query execution with microsecond precision.
- `query_duration_ms` (`UInt64`) — Duration of query execution in milliseconds.
- `read_rows` (`UInt64`) — Total number of rows read from all tables and table functions participated in query. It includes usual subqueries, subqueries for `IN` and `JOIN`. For distributed queries `read_rows` includes the total number of rows read at all replicas. Each replica sends its `read_rows` value, and the server-initiator of the query summarizes all received and local values. The cache volumes do not affect this value.
- `read_bytes` (`UInt64`) — Total number of bytes read from all tables and table functions participated in query. It includes usual subqueries, subqueries for `IN` and `JOIN`. For distributed queries `read_bytes` includes the total number of rows read at all replicas. Each replica sends its `read_bytes` value, and the server-initiator of the query summarizes all received and local values. The cache volumes do not affect this value.

- `written_rows` (`UInt64`) — For `INSERT` queries, the number of written rows. For other queries, the column value is 0.
- `written_bytes` (`UInt64`) — For `INSERT` queries, the number of written bytes. For other queries, the column value is 0.
- `result_rows` (`UInt64`) — Number of rows in a result of the `SELECT` query, or a number of rows in the `INSERT` query.
- `result_bytes` (`UInt64`) — RAM volume in bytes used to store a query result.
- `memory_usage` (`UInt64`) — Memory consumption by the query.
- `current_database` (`String`) — Name of the current database.
- `query` (`String`) — Query string.
- `normalized_query_hash` (`UInt64`) — Identical hash value without the values of literals for similar queries.
- `query_kind` (`LowCardinality(String)`) — Type of the query.
- `databases` (`Array(LowCardinality(String))`) — Names of the databases present in the query.
- `tables` (`Array(LowCardinality(String))`) — Names of the tables present in the query.
- `views` (`Array(LowCardinality(String))`) — Names of the (materialized or live) views present in the query.
- `columns` (`Array(LowCardinality(String))`) — Names of the columns present in the query.
- `projections` (`String`) — Names of the projections used during the query execution.
- `exception_code` (`Int32`) — Code of an exception.
- `exception` (`String`) — Exception message.
- `stack_trace` (`String`) — **Stack trace**. An empty string, if the query was completed successfully.
- `is_initial_query` (`UInt8`) — Query type. Possible values:
  - 1 — Query was initiated by the client.
  - 0 — Query was initiated by another query as part of 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.
- `initial_query_start_time` (`DateTime`) — Initial query starting time (for distributed query execution).
- `initial_query_start_time_microseconds` (`DateTime64`) — Initial query starting time with microseconds precision (for distributed query execution).

- `interface` (`UInt8`) — Interface that the query was initiated from. Possible values:
  - 1 — TCP.
  - 2 — HTTP.
- `os_user` (`String`) — Operating system username who runs `clickhouse-client`.
- `client_hostname` (`String`) — Hostname of the client machine where the `clickhouse-client` or another TCP client is run.
- `client_name` (`String`) — The `clickhouse-client` or another TCP client name.
- `client_revision` (`UInt32`) — Revision of the `clickhouse-client` or another TCP client.
- `client_version_major` (`UInt32`) — Major version of the `clickhouse-client` or another TCP client.
- `client_version_minor` (`UInt32`) — Minor version of the `clickhouse-client` or another TCP client.
- `client_version_patch` (`UInt32`) — Patch component of the `clickhouse-client` or another TCP client version.
- `http_method` (`UInt8`) — HTTP method that initiated the query. Possible values:
  - 0 — The query was launched from the TCP interface.
  - 1 — `GET` method was used.
  - 2 — `POST` method was used.
- `http_user_agent` (`String`) — HTTP header `UserAgent` passed in the HTTP query.
- `http_referer` (`String`) — HTTP header `Referer` passed in the HTTP query (contains an absolute or partial address of the page making the query).
- `forwarded_for` (`String`) — HTTP header `X-Forwarded-For` passed in the HTTP query.
- `quota_key` (`String`) — The quota key specified in the `quotas` setting (see `keyed`).
- `revision` (`UInt32`) — ClickHouse revision.
- `ProfileEvents` (`Map(String, UInt64)`) — ProfileEvents that measure different metrics. The description of them could be found in the table `system.events`
- `Settings` (`Map(String, String)`) — Settings that were changed when the client ran the query. To enable logging changes to settings, set the `log_query_settings` parameter to 1.
- `log_comment` (`String`) — Log comment. It can be set to arbitrary string no longer than `max_query_size`. An empty string if it is not defined.
- `thread_ids` (`Array(UInt64)`) — Thread ids that are participating in query execution.
- `used_aggregate_functions` (`Array(String)`) — Canonical names of `aggregate functions`, which were used during query execution.
- `used_aggregate_function_combinators` (`Array(String)`) — Canonical names of `aggregate functions combinators`, which were used during query execution.
- `used_database_engines` (`Array(String)`) — Canonical names of `database engines`, which were used during query execution.
- `used_data_type_families` (`Array(String)`) — Canonical names of `data type families`, which were used during query execution.

- `used_dictionaries` (`Array(String)`) — Canonical names of `dictionaries`, which were used during query execution.
- `used_formats` (`Array(String)`) — Canonical names of `formats`, which were used during query execution.
- `used_functions` (`Array(String)`) — Canonical names of `functions`, which were used during query execution.
- `used_storages` (`Array(String)`) — Canonical names of `storages`, which were used during query execution.
- `used_table_functions` (`Array(String)`) — Canonical names of `table functions`, which were used during query execution.

## Example

```
SELECT * FROM system.query_log WHERE type = 'QueryFinish' ORDER BY query_start_time DESC LIMIT 1 FORMAT Vertical;
```

Row 1:

```

type:          QueryFinish
event_date:    2021-07-28
event_time:    2021-07-28 13:46:56
event_time_microseconds: 2021-07-28 13:46:56.719791
query_start_time: 2021-07-28 13:46:56
query_start_time_microseconds: 2021-07-28 13:46:56.704542
query_duration_ms: 14
read_rows:     8393
read_bytes:   374325
written_rows: 0
written_bytes: 0
result_rows:  4201
result_bytes: 153024
memory_usage: 4714038
current_database: default
query:          SELECT DISTINCT arrayJoin(extractAll(name, '[\\w_]{2,}') AS res FROM (SELECT name FROM system.functions UNION ALL SELECT name FROM system.table_engines UNION ALL SELECT name FROM system.formats UNION ALL SELECT name FROM system.table_functions UNION ALL SELECT name FROM system.data_type_families UNION ALL SELECT name FROM system.merge_tree_settings UNION ALL SELECT name FROM system.settings UNION ALL SELECT cluster FROM system.clusters UNION ALL SELECT macro FROM system.macros UNION ALL SELECT policy_name FROM system.storage_policies UNION ALL SELECT concat(func.name, comb.name) FROM system.functions AS func CROSS JOIN system.aggregate_function_combinators AS comb WHERE is_aggregate UNION ALL SELECT name FROM system.databases LIMIT 10000 UNION ALL SELECT DISTINCT name FROM system.tables LIMIT 10000 UNION ALL SELECT DISTINCT name FROM system.dictionaries LIMIT 10000 UNION ALL SELECT DISTINCT name FROM system.columns LIMIT 10000) WHERE notEmpty(res)
normalized_query_hash: 6666026786019643712
query_kind:      Select
databases:      ['system']
tables:          ['system.aggregate_function_combinators','system.clusters','system.columns','system.data_type_families','system.databases','system.dictionaries','system.formats','system.functions','system.macros','system.merge_tree_settings','system.settings','system.storage_policies','system.table_engines','system.table_functions','system.tables']
columns:         ['system.aggregate_function_combinators.name','system.clusters.cluster','system.columns.name','system.data_type_families.name','system.databases.name','system.dictionaries.name','system.formats.name','system.functions.is_aggregate','system.functions.name','system.macros.macro','system.merge_tree_settings.name','system.settings.name','system.storage_policies.policy_name','system.table_engines.name','system.table_functions.name','system.tables.name']
projections:    []
exception_code: 0
exception:
stack_trace:
is_initial_query: 1
user:            default
query_id:        a3361f6e-a1fd-4d54-9f6f-f93a08bab0bf
address:         ::ffff:127.0.0.1
port:            51006
initial_user:   default
initial_query_id: a3361f6e-a1fd-4d54-9f6f-f93a08bab0bf
initial_address: ::ffff:127.0.0.1
initial_port:   51006
initial_query_start_time: 2021-07-28 13:46:56

```

```

initial_query_start_time_microseconds: 2021-07-28 13:46:56.704542
interface: 1
os_user:
client_hostname:
client_name: ClickHouse client
client_revision: 54449
client_version_major: 21
client_version_minor: 8
client_version_patch: 0
http_method: 0
http_user_agent:
http_referer:
forwarded_for:
quota_key:
revision: 54453
log_comment:
thread_ids: [5058,22097,22110,22094]
ProfileEvents.Names:
['Query','SelectQuery','ArenaAllocChunks','ArenaAllocBytes','FunctionExecute','NetworkSendElapsedMicroseconds','SelectedRows','SelectedBytes','ContextLock','RWLockAcquiredReadLocks','RealTimeMicroseconds','UserTimeMicroseconds','SystemTimeMicroseconds','SoftPageFaults','OSCPUWaitMicroseconds','OSCPUVirtualTimeMicroseconds','OSWriteBytes','OSWriteChars']
ProfileEvents.Values:
[1,1,39,352256,64,360,8393,374325,412,440,34480,13108,4723,671,19,17828,8192,10240]
Settings.Names: ['load_balancing','max_memory_usage']
Settings.Values: ['random','10000000000']
used_aggregate_functions: []
used_aggregate_function_combinators: []
used_database_engines: []
used_data_type_families: ['UInt64','UInt8','Nullable','String','date']
used_dictionaries: []
used_formats: []
used_functions: ['concat','notEmpty','extractAll']
used_storages: []
used_table_functions: []

```

## See Also

- [system.query\\_thread\\_log](#) — This table contains information about each query execution thread.
- [system.query\\_views\\_log](#) — This table contains information about each view executed during a query.

## system.query\_thread\_log

Contains information about threads that execute queries, for example, thread name, thread start time, duration of query processing.

To start logging:

1. Configure parameters in the [query\\_thread\\_log](#) section.
2. Set [log\\_query\\_threads](#) to 1.

The flushing period of data is set in [flush\\_interval\\_milliseconds](#) parameter of the [query\\_thread\\_log](#) server settings section. To force flushing, use the [SYSTEM FLUSH LOGS](#) query.

ClickHouse does not delete data from the table automatically. See [Introduction](#) for more details.

You can use the [log\\_queries\\_probability](#) setting to reduce the number of queries, registered in the [query\\_thread\\_log](#) table.

Columns:

- [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.

- `event_time_microseconds` (`DateTime`) — The date and time when the thread has finished execution of the query with microseconds precision.
- `query_start_time` (`DateTime`) — Start time of query execution.
- `query_start_time_microseconds` (`DateTime64`) — Start time of query execution with microsecond precision.
- `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` queries, the number of written rows. For other queries, the column value is 0.
- `written_bytes` (`UInt64`) — For `INSERT` queries, the number of written bytes. For other queries, the column value is 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.
- `thread_id` (`Int32`) — 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 `clickhouse-client`.

- `client_hostname` (`String`) — Hostname of the client machine where the `clickhouse-client` or another TCP client is run.
- `client_name` (`String`) — The `clickhouse-client` or another TCP client name.
- `client_revision` (`UInt32`) — Revision of the `clickhouse-client` or another TCP client.
- `client_version_major` (`UInt32`) — Major version of the `clickhouse-client` or another TCP client.
- `client_version_minor` (`UInt32`) — Minor version of the `clickhouse-client` or another TCP client.
- `client_version_patch` (`UInt32`) — Patch component of the `clickhouse-client` or another TCP client version.
- `http_method` (`UInt8`) — HTTP method that initiated the query. Possible values:
  - 0 — The query was launched from the TCP interface.
  - 1 — `GET` method was used.
  - 2 — `POST` method was used.
- `http_user_agent` (`String`) — The `UserAgent` header passed in the HTTP request.
- `quota_key` (`String`) — The “quota key” specified in the `quotas` setting (see `keyed`).
- `revision` (`UInt32`) — ClickHouse revision.
- `ProfileEvents` (`Map(String, UInt64)`) — ProfileEvents that measure different metrics for this thread. The description of them could be found in the table `system.events`.

## Example

```
SELECT * FROM system.query_thread_log LIMIT 1 \G
```

Row 1:

```
event_date:          2020-09-11
event_time:          2020-09-11 10:08:17
event_time_microseconds: 2020-09-11 10:08:17.134042
query_start_time:    2020-09-11 10:08:17
query_start_time_microseconds: 2020-09-11 10:08:17.063150
query_duration_ms:   70
read_rows:           0
read_bytes:          0
written_rows:        1
written_bytes:       12
memory_usage:        4300844
peak_memory_usage:   4300844
thread_name:         TCPHandler
thread_id:           638133
master_thread_id:    638133
query:               INSERT INTO test1 VALUES
is_initial_query:   1
user:                default
query_id:            50a320fd-85a8-49b8-8761-98a86bcbacef
address:             ::ffff:127.0.0.1
port:                33452
initial_user:        default
initial_query_id:   50a320fd-85a8-49b8-8761-98a86bcbacef
initial_address:    ::ffff:127.0.0.1
initial_port:        33452
interface:          1
os_user:             bharatnc
client_hostname:    tower
client_name:         ClickHouse
client_revision:    54437
client_version_major: 20
client_version_minor: 7
client_version_patch: 2
http_method:         0
http_user_agent:
quota_key:
revision:            54440
ProfileEvents:
{'Query':1,'SelectQuery':1,'ReadCompressedBytes':36,'CompressedReadBufferBlocks':1,'CompressedReadBufferBytes':10,'IOBufferAllocs':1,'IOBufferAllocBytes':89,'ContextLock':15,'RWLockAcquiredReadLocks':1}
```

## See Also

- [system.query\\_log](#) — Description of the `query_log` system table which contains common information about queries execution.
- [system.query\\_views\\_log](#) — This table contains information about each view executed during a query.

## system.query\_views\_log

Contains information about the dependent views executed when running a query, for example, the view type or the execution time.

To start logging:

1. Configure parameters in the `query_views_log` section.
2. Set `log_query_views` to 1.

The flushing period of data is set in `flush_interval_milliseconds` parameter of the `query_views_log` server settings section. To force flushing, use the [SYSTEM FLUSH LOGS](#) query.

ClickHouse does not delete data from the table automatically. See [Introduction](#) for more details.

You can use the `log_queries_probability` setting to reduce the number of queries, registered in the `query_views_log` table.

Columns:

- `event_date` (**Date**) — The date when the last event of the view happened.
- `event_time` (**DateTime**) — The date and time when the view finished execution.
- `event_time_microseconds` (**DateTime**) — The date and time when the view finished execution with microseconds precision.
- `view_duration_ms` (**UInt64**) — Duration of view execution (sum of its stages) in milliseconds.
- `initial_query_id` (**String**) — ID of the initial query (for distributed query execution).
- `view_name` (**String**) — Name of the view.
- `view_uuid` (**UUID**) — UUID of the view.
- `view_type` (**Enum8**) — Type of the view. Values:
  - `'Default'` = 1 — **Default views**. Should not appear in this log.
  - `'Materialized'` = 2 — **Materialized views**.
  - `'Live'` = 3 — **Live views**.
- `view_query` (**String**) — The query executed by the view.
- `view_target` (**String**) — The name of the view target table.
- `read_rows` (**UInt64**) — Number of read rows.
- `read_bytes` (**UInt64**) — Number of read bytes.
- `written_rows` (**UInt64**) — Number of written rows.
- `written_bytes` (**UInt64**) — Number of written bytes.
- `peak_memory_usage` (**Int64**) — The maximum difference between the amount of allocated and freed memory in context of this view.
- `ProfileEvents` (**Map(String, UInt64)**) — ProfileEvents that measure different metrics. The description of them could be found in the table [system.events](#).
- `status` (**Enum8**) — Status of the view. Values:
  - `'QueryStart'` = 1 — Successful start the view execution. Should not appear.
  - `'QueryFinish'` = 2 — Successful end of the view execution.
  - `'ExceptionBeforeStart'` = 3 — Exception before the start of the view execution.
  - `'ExceptionWhileProcessing'` = 4 — Exception during the view execution.
- `exception_code` (**Int32**) — Code of an exception.
- `exception` (**String**) — Exception message.
- `stack_trace` (**String**) — **Stack trace**. An empty string, if the query was completed successfully.

## Example

Query:

```
SELECT * FROM system.query_views_log LIMIT 1 \G;
```

Result:

```
Row 1:  
event_date: 2021-06-22  
event_time: 2021-06-22 13:23:07  
event_time_microseconds: 2021-06-22 13:23:07.738221  
view_duration_ms: 0  
initial_query_id: c3a1ac02-9cad-479b-af54-9e9c0a7afd70  
view_name: default.matview_inner  
view_uuid: 00000000-0000-0000-0000-000000000000  
view_type: Materialized  
view_query: SELECT * FROM default.table_b  
view_target: default.`.inner.matview_inner`  
read_rows: 4  
read_bytes: 64  
written_rows: 2  
written_bytes: 32  
peak_memory_usage: 4196188  
ProfileEvents:  
{'FileOpen':2,'WriteBufferFromFileDescriptorWrite':2,'WriteBufferFromFileDescriptorWriteBytes':187,'IOBufferAllocs':3  
'IOBufferAllocBytes':3145773,'FunctionExecute':3,'DiskWriteElapsedMicroseconds':13,'InsertedRows':2,'InsertedBytes  
'SoftPageFaults':4,'OSReadChars':463}  
status: QueryFinish  
exception_code: 0  
exception:  
stack_trace:
```

## See Also

- [system.query\\_log](#) — Description of the `query_log` system table which contains common information about queries execution.
- [system.query\\_thread\\_log](#) — This table contains information about each query execution thread.

## system.quota\_limits

Contains information about maximums for all intervals of all quotas. Any number of rows or zero can correspond to one quota.

Columns:

- quota\_name (**String**) — Quota name.
- duration (**UInt32**) — Length of the time interval for calculating resource consumption, in seconds.
- is\_randomized\_interval (**UInt8**) — Logical value. It shows whether the interval is randomized. Interval always starts at the same time if it is not randomized. For example, an interval of 1 minute always starts at an integer number of minutes (i.e. it can start at 11:20:00, but it never starts at 11:20:01), an interval of one day always starts at midnight UTC. If interval is randomized, the very first interval starts at random time, and subsequent intervals starts one by one. Values:
  - **0** — Interval is not randomized.
  - **1** — Interval is randomized.
- max\_queries (**Nullable(UInt64)**) — Maximum number of queries.
- max\_query\_selects (**Nullable(UInt64)**) — Maximum number of select queries.
- max\_query\_inserts (**Nullable(UInt64)**) — Maximum number of insert queries.
- max\_errors (**Nullable(UInt64)**) — Maximum number of errors.
- max\_result\_rows (**Nullable(UInt64)**) — Maximum number of result rows.
- max\_result\_bytes (**Nullable(UInt64)**) — Maximum number of RAM volume in bytes used to store a queries result.
- max\_read\_rows (**Nullable(UInt64)**) — Maximum number of rows read from all tables and table functions participated in queries.

- `max_read_bytes` (`Nullable(UInt64)`) — Maximum number of bytes read from all tables and table functions participated in queries.
  - `max_execution_time` (`Nullable(Float64)`) — Maximum of the query execution time, in seconds.
- 

## system.quota\_usage

Quota usage by the current user: how much is used and how much is left.

Columns:

- `quota_name` (`String`) — Quota name.
- `quota_key` (`String`) — Key value. For example, if keys = [ip address], `quota_key` may have a value '192.168.1.1'.
- `start_time` (`Nullable(DateTime)`) — Start time for calculating resource consumption.
- `end_time` (`Nullable(DateTime)`) — End time for calculating resource consumption.
- `duration` (`Nullable(UInt64)`) — Length of the time interval for calculating resource consumption, in seconds.
- `queries` (`Nullable(UInt64)`) — The total number of requests on this interval.
- `query_selects` (`Nullable(UInt64)`) — The total number of select requests on this interval.
- `query_inserts` (`Nullable(UInt64)`) — The total number of insert requests on this interval.
- `max_queries` (`Nullable(UInt64)`) — Maximum number of requests.
- `errors` (`Nullable(UInt64)`) — The number of queries that threw an exception.
- `max_errors` (`Nullable(UInt64)`) — Maximum number of errors.
- `result_rows` (`Nullable(UInt64)`) — The total number of rows given as a result.
- `max_result_rows` (`Nullable(UInt64)`) — Maximum number of result rows.
- `result_bytes` (`Nullable(UInt64)`) — RAM volume in bytes used to store a queries result.
- `max_result_bytes` (`Nullable(UInt64)`) — Maximum RAM volume used to store a queries result, in bytes.
- `read_rows` (`Nullable(UInt64)`) — The total number of source rows read from tables for running the query on all remote servers.
- `max_read_rows` (`Nullable(UInt64)`) — Maximum number of rows read from all tables and table functions participated in queries.
- `read_bytes` (`Nullable(UInt64)`) — The total number of bytes read from all tables and table functions participated in queries.
- `max_read_bytes` (`Nullable(UInt64)`) — Maximum of bytes read from all tables and table functions.
- `execution_time` (`Nullable(Float64)`) — The total query execution time, in seconds (wall time).
- `max_execution_time` (`Nullable(Float64)`) — Maximum of query execution time.

## See Also

- [SHOW QUOTA](#)
- 

## system.quotas

Contains information about [quotas](#).

Columns:

- `name` (**String**) — Quota name.
- `id` (**UUID**) — Quota ID.
- `storage`(**String**) — Storage of quotas. Possible value: “users.xml” if a quota configured in the users.xml file, “disk” if a quota configured by an SQL-query.
- `keys` (**Array(Enum8)**) — Key specifies how the quota should be shared. If two connections use the same quota and key, they share the same amounts of resources. Values:
  - `[]` — All users share the same quota.
  - `['user_name']` — Connections with the same user name share the same quota.
  - `['ip_address']` — Connections from the same IP share the same quota.
  - `['client_key']` — Connections with the same key share the same quota. A key must be explicitly provided by a client. When using `clickhouse-client`, pass a key value in the `--quota_key` parameter, or use the `quota_key` parameter in the client configuration file. When using HTTP interface, use the `X-ClickHouse-Quota` header.
  - `['user_name', 'client_key']` — Connections with the same `client_key` share the same quota. If a key isn’t provided by a client, the quota is tracked for `user_name`.
  - `['client_key', 'ip_address']` — Connections with the same `client_key` share the same quota. If a key isn’t provided by a client, the quota is tracked for `ip_address`.
- `durations` (**Array(UInt64)**) — Time interval lengths in seconds.
- `apply_to_all` (**UInt8**) — Logical value. It shows which users the quota is applied to. Values:
  - `0` — The quota applies to users specified in the `apply_to_list`.
  - `1` — The quota applies to all users except those listed in `apply_to_except`.
- `apply_to_list` (**Array(String)**) — List of user names/**roles** that the quota should be applied to.
- `apply_to_except` (**Array(String)**) — List of user names/**roles** that the quota should not apply to.

## See Also

- [SHOW QUOTAS](#)

## system.quotas\_usage

Quota usage by all users.

Columns:

- quota\_name ([String](#)) — Quota name.
- quota\_key ([String](#)) — Key value.
- is\_current ([UInt8](#)) — Quota usage for current user.
- start\_time ([Nullable\(DateTime\)](#)) — Start time for calculating resource consumption.
- end\_time ([Nullable\(DateTime\)](#)) — End time for calculating resource consumption.
- duration ([Nullable\(UInt32\)](#)) — Length of the time interval for calculating resource consumption, in seconds.
- queries ([Nullable\(UInt64\)](#)) — The total number of requests in this interval.
- max\_queries ([Nullable\(UInt64\)](#)) — Maximum number of requests.
- query\_selects ([Nullable\(UInt64\)](#)) — The total number of select requests in this interval.
- max\_query\_selects ([Nullable\(UInt64\)](#)) — Maximum number of select requests.
- query\_inserts ([Nullable\(UInt64\)](#)) — The total number of insert requests in this interval.
- max\_query\_inserts ([Nullable\(UInt64\)](#)) — Maximum number of insert requests.
- errors ([Nullable\(UInt64\)](#)) — The number of queries that threw an exception.
- max\_errors ([Nullable\(UInt64\)](#)) — Maximum number of errors.
- result\_rows ([Nullable\(UInt64\)](#)) — The total number of rows given as a result.
- max\_result\_rows ([Nullable\(UInt64\)](#)) — Maximum of source rows read from tables.
- result\_bytes ([Nullable\(UInt64\)](#)) — RAM volume in bytes used to store a queries result.
- max\_result\_bytes ([Nullable\(UInt64\)](#)) — Maximum RAM volume used to store a queries result, in bytes.
- read\_rows ([Nullable\(UInt64\)](#)) — The total number of source rows read from tables for running the query on all remote servers.
- max\_read\_rows ([Nullable\(UInt64\)](#)) — Maximum number of rows read from all tables and table functions participated in queries.
- read\_bytes ([Nullable\(UInt64\)](#)) — The total number of bytes read from all tables and table functions participated in queries.
- max\_read\_bytes ([Nullable\(UInt64\)](#)) — Maximum of bytes read from all tables and table functions.
- execution\_time ([Nullable\(Float64\)](#)) — The total query execution time, in seconds (wall time).
- max\_execution\_time ([Nullable\(Float64\)](#)) — Maximum of query execution time.

## See Also

- [SHOW QUOTA](#)

## system.replicas

Contains information and status for replicated tables residing on the local server.

This table can be used for monitoring. The table contains a row for every Replicated\* table.

Example:

```
SELECT *
FROM system.replicas
WHERE table = 'test_table'
FORMAT Vertical
```

Query id: dc6dcacb-dc28-4df9-ae27-4354f5b3b13e

Row 1:

```
database:          db
table:            test_table
engine:           ReplicatedMergeTree
is_leader:        1
can_become_leader: 1
is_readonly:      0
is_session_expired: 0
future_parts:    0
parts_to_check:  0
zookeeper_path: /test/test_table
replica_name:    r1
replica_path:   /test/test_table/replicas/r1
columns_version: -1
queue_size:      27
inserts_in_queue: 27
merges_in_queue:  0
part_mutations_in_queue: 0
queue_oldest_time: 2021-10-12 14:48:48
inserts_oldest_time: 2021-10-12 14:48:48
merges_oldest_time: 1970-01-01 03:00:00
part_mutations_oldest_time: 1970-01-01 03:00:00
oldest_part_to_get: 1_17_17_0
oldest_part_to_merge_to:
oldest_part_to_mutate_to:
log_max_index:   206
log_pointer:     207
last_queue_update: 2021-10-12 14:50:08
absolute_delay:  99
total_replicas:  5
active_replicas: 5
last_queue_update_exception:
zookeeper_exception:
replica_is_active: {'r1':1,'r2':1}
```

Columns:

- **database** (String) - Database name
- **table** (String) - Table name
- **engine** (String) - Table engine name
- **is\_leader** (UInt8) - Whether the replica is the leader.  
Multiple replicas can be leaders at the same time. A replica can be prevented from becoming a leader using the `merge_tree` setting `replicated_can_become_leader`. The leaders are responsible for scheduling background merges.  
Note that writes can be performed to any replica that is available and has a session in ZK, regardless of whether it is a leader.
- **can\_become\_leader** (UInt8) - Whether the replica can be a leader.
- **is\_READONLY** (UInt8) - Whether the replica is in read-only mode.  
This mode is turned on if the config does not have sections with ZooKeeper, if an unknown error occurred when reinitializing sessions in ZooKeeper, and during session reinitialization in ZooKeeper.
- **is\_SESSION\_EXPIRED** (UInt8) - the session with ZooKeeper has expired. Basically the same as `is_READONLY`.
- **future\_parts** (UInt32) - The number of data parts that will appear as the result of INSERTs or merges that haven't been done yet.
- **parts\_to\_check** (UInt32) - The number of data parts in the queue for verification. A part is put in the verification queue if there is suspicion that it might be damaged.

- `zookeeper_path` (`String`) - Path to table data in ZooKeeper.
- `replica_name` (`String`) - Replica name in ZooKeeper. Different replicas of the same table have different names.
- `replica_path` (`String`) - Path to replica data in ZooKeeper. The same as concatenating '`zookeeper_path/replicas/replica_path`'.
- `columns_version` (`Int32`) - Version number of the table structure. Indicates how many times ALTER was performed. If replicas have different versions, it means some replicas haven't made all of the ALTERs yet.
- `queue_size` (`UInt32`) - Size of the queue for operations waiting to be performed. Operations include inserting blocks of data, merges, and certain other actions. It usually coincides with `future_parts`.
- `inserts_in_queue` (`UInt32`) - Number of inserts of blocks of data that need to be made. Insertions are usually replicated fairly quickly. If this number is large, it means something is wrong.
- `merges_in_queue` (`UInt32`) - The number of merges waiting to be made. Sometimes merges are lengthy, so this value may be greater than zero for a long time.
- `part_mutations_in_queue` (`UInt32`) - The number of mutations waiting to be made.
- `queue_oldest_time` (`DateTime`) - If `queue_size` greater than 0, shows when the oldest operation was added to the queue.
- `inserts_oldest_time` (`DateTime`) - See `queue_oldest_time`
- `merges_oldest_time` (`DateTime`) - See `queue_oldest_time`
- `part_mutations_oldest_time` (`DateTime`) - See `queue_oldest_time`

The next 4 columns have a non-zero value only where there is an active session with ZK.

- `log_max_index` (`UInt64`) - Maximum entry number in the log of general activity.
- `log_pointer` (`UInt64`) - Maximum entry number in the log of general activity that the replica copied to its execution queue, plus one. If `log_pointer` is much smaller than `log_max_index`, something is wrong.
- `last_queue_update` (`DateTime`) - When the queue was updated last time.
- `absolute_delay` (`UInt64`) - How big lag in seconds the current replica has.
- `total_replicas` (`UInt8`) - The total number of known replicas of this table.
- `active_replicas` (`UInt8`) - The number of replicas of this table that have a session in ZooKeeper (i.e., the number of functioning replicas).
- `last_queue_update_exception` (`String`) - When the queue contains broken entries. Especially important when ClickHouse breaks backward compatibility between versions and log entries written by newer versions aren't parseable by old versions.
- `zookeeper_exception` (`String`) - The last exception message, got if the error happened when fetching the info from ZooKeeper.
- `replica_is_active` (`Map(String, UInt8)`) — Map between replica name and is replica active.

If you request all the columns, the table may work a bit slowly, since several reads from ZooKeeper are made for each row.

If you do not request the last 4 columns (`log_max_index`, `log_pointer`, `total_replicas`, `active_replicas`), the table works quickly.

For example, you can check that everything is working correctly like this:

```
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
```

If this query does not return anything, it means that everything is fine.

## system.replicated\_fetches

Contains information about currently running background fetches.

Columns:

- `database` (**String**) — Name of the database.
- `table` (**String**) — Name of the table.
- `elapsed` (**Float64**) — The time elapsed (in seconds) since showing currently running background fetches started.
- `progress` (**Float64**) — The percentage of completed work from 0 to 1.
- `result_part_name` (**String**) — The name of the part that will be formed as the result of showing currently running background fetches.
- `result_part_path` (**String**) — Absolute path to the part that will be formed as the result of showing currently running background fetches.
- `partition_id` (**String**) — ID of the partition.
- `total_size_bytes_compressed` (**UInt64**) — The total size (in bytes) of the compressed data in the result part.
- `bytes_read_compressed` (**UInt64**) — The number of compressed bytes read from the result part.
- `source_replica_path` (**String**) — Absolute path to the source replica.
- `source_replica_hostname` (**String**) — Hostname of the source replica.
- `source_replica_port` (**UInt16**) — Port number of the source replica.

- `interserver_scheme` ([String](#)) — Name of the interserver scheme.
- `URI` ([String](#)) — Uniform resource identifier.
- `to_detached` ([UInt8](#)) — The flag indicates whether the currently running background fetch is being performed using the `TO DETACHED` expression.
- `thread_id` ([UInt64](#)) — Thread identifier.

## Example

```
SELECT * FROM system.replicated_fetches LIMIT 1 FORMAT Vertical;
```

Row 1:

```
database:          default
table:             t
elapsed:           7.243039876
progress:          0.41832135995612835
result_part_name: all_0_0_0
result_part_path: /var/lib/clickhouse/store/700/70080a04-b2de-4adf-9fa5-9ea210e81766/all_0_0_0/
partition_id:      all
total_size_bytes_compressed: 1052783726
bytes_read_compressed: 440401920
source_replica_path: /clickhouse/test/t/replicas/1
source_replica_hostname: node1
source_replica_port: 9009
interserver_scheme: http
URI:               http://node1:9009/?endpoint=DataPartsExchange%3A%2Fclickhouse%2Ftest%2Ft%2Freplicas%2F1&part=all_0_0_0&client_protocol_version=4&compress=false
to_detached:       0
thread_id:         54
```

## See Also

- [Managing ReplicatedMergeTree Tables](#)

# system.replication\_queue

Contains information about tasks from replication queues stored in ZooKeeper for tables in the `ReplicatedMergeTree` family.

Columns:

- `database` ([String](#)) — Name of the database.
- `table` ([String](#)) — Name of the table.
- `replica_name` ([String](#)) — Replica name in ZooKeeper. Different replicas of the same table have different names.
- `position` ([UInt32](#)) — Position of the task in the queue.
- `node_name` ([String](#)) — Node name in ZooKeeper.

- `type` (**String**) — Type of the task in the queue, one of:
  - `GET_PART` — Get the part from another replica.
  - `ATTACH_PART` — Attach the part, possibly from our own replica (if found in the `detached` folder). You may think of it as a `GET_PART` with some optimizations as they're nearly identical.
  - `MERGE_PARTS` — Merge the parts.
  - `DROP_RANGE` — Delete the parts in the specified partition in the specified number range.
  - `CLEAR_COLUMN` — NOTE: Deprecated. Drop specific column from specified partition.
  - `CLEAR_INDEX` — NOTE: Deprecated. Drop specific index from specified partition.
  - `REPLACE_RANGE` — Drop a certain range of parts and replace them with new ones.
  - `MUTATE_PART` — Apply one or several mutations to the part.
  - `ALTER_METADATA` — Apply alter modification according to global `/metadata` and `/columns` paths.
- `create_time` (**Datetime**) — Date and time when the task was submitted for execution.
- `required_quorum` (**UInt32**) — The number of replicas waiting for the task to complete with confirmation of completion. This column is only relevant for the `GET_PARTS` task.
- `source_replica` (**String**) — Name of the source replica.
- `new_part_name` (**String**) — Name of the new part.
- `parts_to_merge` (**Array (String)**) — Names of parts to merge or update.
- `is_detach` (**UInt8**) — The flag indicates whether the `DETACH_PARTS` task is in the queue.
- `is_currently_executing` (**UInt8**) — The flag indicates whether a specific task is being performed right now.
- `num_tries` (**UInt32**) — The number of failed attempts to complete the task.
- `last_exception` (**String**) — Text message about the last error that occurred (if any).
- `last_attempt_time` (**Datetime**) — Date and time when the task was last attempted.
- `num_postponed` (**UInt32**) — The number of postponed tasks.
- `postpone_reason` (**String**) — The reason why the task was postponed.
- `last_postpone_time` (**Datetime**) — Date and time when the task was last postponed.
- `merge_type` (**String**) — Type of the current merge. Empty if it's a mutation.

## Example

```
SELECT * FROM system.replication_queue LIMIT 1 FORMAT Vertical;
```

Row 1:

```
database:      merge
table:        visits_v2
replica_name:  mtgiga001-1t.metrika.yandex.net
position:     15
node_name:    queue-0009325559
type:         MERGE_PARTS
create_time:   2020-12-07 14:04:21
required_quorum: 0
source_replica: mtgiga001-1t.metrika.yandex.net
new_part_name:  20201130_121373_121384_2
parts_to_merge: ['20201130_121373_121378_1','20201130_121379_121379_0','20201130_121380_121380_0','20201130_121381_121381_0','20201130_121382_121382_0','20201130_121383_121383_0','20201130_121384_121384_0']
is_detach:    0
is_currently_executing: 0
num_tries:    36
last_exception: Code: 226, e.displayText() = DB::Exception: Marks file '/opt/clickhouse/data/merge/visits_v2/tmp_fetch_20201130_121373_121384_2/CounterID.mrk' does not exist (version 20.8.7.15 (official build))
last_attempt_time: 2020-12-08 17:35:54
num_postponed: 0
postpone_reason:
last_postpone_time: 1970-01-01 03:00:00
```

## See Also

- [Managing ReplicatedMergeTree Tables](#)

## system.role\_grants

Contains the role grants for users and roles. To add entries to this table, use `GRANT role TO user`.

Columns:

- `user_name` (`Nullable(String)`) — User name.
- `role_name` (`Nullable(String)`) — Role name.
- `granted_role_name` (`String`) — Name of role granted to the `role_name` role. To grant one role to another one use `GRANT role1 TO role2`.
- `granted_role_is_default` (`UInt8`) — Flag that shows whether `granted_role` is a default role. Possible values:
  - 1 — `granted_role` is a default role.
  - 0 — `granted_role` is not a default role.
- `with_admin_option` (`UInt8`) — Flag that shows whether `granted_role` is a role with `ADMIN OPTION` privilege. Possible values:
  - 1 — The role has `ADMIN OPTION` privilege.
  - 0 — The role without `ADMIN OPTION` privilege.

## system.roles

Contains information about configured `roles`.

Columns:

- `name` (`String`) — Role name.

- `id` ([UUID](#)) — Role ID.
- `storage` ([String](#)) — Path to the storage of roles. Configured in the `access_control_path` parameter.

## See Also

- [SHOW ROLES](#)

# system.row\_policies

Contains filters for one particular table, as well as a list of roles and/or users which should use this row policy.

Columns:

- `name` ([String](#)) — Name of a row policy.
- `short_name` ([String](#)) — Short name of a row policy. Names of row policies are compound, for example: myfilter ON mydb.mytable. Here "myfilter ON mydb.mytable" is the name of the row policy, "myfilter" is its short name.
- `database` ([String](#)) — Database name.
- `table` ([String](#)) — Table name.
- `id` ([UUID](#)) — Row policy ID.
- `storage` ([String](#)) — Name of the directory where the row policy is stored.
- `select_filter` ([Nullable\(String\)](#)) — Condition which is used to filter rows.
- `is_restrictive` ([UInt8](#)) — Shows whether the row policy restricts access to rows, see [CREATE ROW POLICY](#). Value:
  - `0` — The row policy is defined with `AS PERMISSIVE` clause.
  - `1` — The row policy is defined with `AS RESTRICTIVE` clause.
- `apply_to_all` ([UInt8](#)) — Shows that the row policies set for all roles and/or users.
- `apply_to_list` ([Array\(String\)](#)) — List of the roles and/or users to which the row policies is applied.
- `apply_to_except` ([Array\(String\)](#)) — The row policies is applied to all roles and/or users excepting of the listed ones.

## See Also

- [SHOW POLICIES](#)

# system.settings

Contains information about session settings for current user.

Columns:

- `name` ([String](#)) — Setting name.
- `value` ([String](#)) — Setting value.
- `changed` ([UInt8](#)) — Shows whether a setting is changed from its default value.

- `description` ([String](#)) — Short setting description.
- `min` ([Nullable\(String\)](#)) — Minimum value of the setting, if any is set via [constraints](#). If the setting has no minimum value, contains [NULL](#).
- `max` ([Nullable\(String\)](#)) — Maximum value of the setting, if any is set via [constraints](#). If the setting has no maximum value, contains [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.

## Example

The following example shows how to get information about settings which name contains `min_i`.

```
SELECT *
FROM system.settings
WHERE name LIKE '%min_i%'
```

name	value	changed	description
min	max	readonly	
<code>  min_insert_block_size_rows</code> in rows, if blocks are not big enough.	<code>  1048576</code>	<code>  0</code>	<code>  Squash blocks passed to INSERT query to specified size</code> <code>  NULL   NULL   0  </code>
<code>  min_insert_block_size_bytes</code> in bytes, if blocks are not big enough.	<code>  268435456</code>	<code>  0</code>	<code>  Squash blocks passed to INSERT query to specified size</code> <code>  NULL   NULL   0  </code>
<code>  read_backoff_min_interval_between_events_ms</code> case of slow reads. Do not pay attention to the event, if the previous one has passed less than a certain amount of time.	<code>  1000</code>	<code>  0</code>	<code>  Settings to reduce the number of threads in</code> <code>  NULL   NULL   0  </code>

Using of `WHERE changed` can be useful, for example, when you want to check:

- Whether settings in configuration files are loaded correctly and are in use.
- Settings that changed in the current session.

```
SELECT * FROM system.settings WHERE changed AND name='load_balancing'
```

## See also

- [Settings](#)
- [Permissions for Queries](#)
- [Constraints on Settings](#)
- [SHOW SETTINGS statement](#)

## system.settings\_profile\_elements

Describes the content of the settings profile:

- Constraints.
- Roles and users that the setting applies to.

- Parent settings profiles.

Columns:

- `profile_name` (`Nullable(String)`) — Setting profile name.
- `user_name` (`Nullable(String)`) — User name.
- `role_name` (`Nullable(String)`) — Role name.
- `index` (`UInt64`) — Sequential number of the settings profile element.
- `setting_name` (`Nullable(String)`) — Setting name.
- `value` (`Nullable(String)`) — Setting value.
- `min` (`Nullable(String)`) — The minimum value of the setting. `NULL` if not set.
- `max` (`Nullable(String)`) — The maximum value of the setting. `NULL` if not set.
- `readonly` (`Nullable(UInt8)`) — Profile that allows only read queries.
- `inherit_profile` (`Nullable(String)`) — A parent profile for this setting profile. `NULL` if not set. Setting profile will inherit all the settings' values and constraints (`min`, `max`, `readonly`) from its parent profiles.

## system.settings\_profiles

Contains properties of configured setting profiles.

Columns:

- `name` (`String`) — Setting profile name.
- `id` (`UUID`) — Setting profile ID.
- `storage` (`String`) — Path to the storage of setting profiles. Configured in the `access_control_path` parameter.
- `num_elements` (`UInt64`) — Number of elements for this profile in the `system.settings_profile_elements` table.
- `apply_to_all` (`UInt8`) — Shows that the settings profile set for all roles and/or users.
- `apply_to_list` (`Array(String)`) — List of the roles and/or users to which the setting profile is applied.
- `apply_to_except` (`Array(String)`) — The setting profile is applied to all roles and/or users excepting of the listed ones.

## See Also

- [SHOW PROFILES](#)

## system.stack\_trace

Contains stack traces of all server threads. Allows developers to introspect the server state.

To analyze stack frames, use the `addressToLine`, `addressToSymbol` and demangle [introspection functions](#).

Columns:

- `thread_name` (`String`) — Thread name.
- `thread_id` (`UInt64`) — Thread identifier.

- `query_id` (**String**) — Query identifier that can be used to get details about a query that was running from the `query_log` system table.
- `trace` (**Array(UInt64)**) — A **stack trace** which represents a list of physical addresses where the called methods are stored.

## Example

Enabling introspection functions:

```
SET allow_introspection_functions = 1;
```

Getting symbols from ClickHouse object files:

```
WITH arrayMap(x -> demangle(addressToSymbol(x)), trace) AS all SELECT thread_name, thread_id, query_id,
arrayStringConcat(all, '\n') AS res FROM system.stack_trace LIMIT 1 \G;
```

Row 1:

```
thread_name: clickhouse-serv

thread_id: 686
query_id: 1a11f70b-626d-47c1-b948-f9c7b206395d
res:    sigqueue
DB::StorageSystemStackTrace::fillData(std::__1::vector<COW<DB::IColumn>::mutable_ptr<DB::IColumn>,
std::__1::allocator<COW<DB::IColumn>::mutable_ptr<DB::IColumn> >>, DB::Context const&, DB::SelectQueryInfo const&) const
DB::IStorageSystemOneBlock<DB::StorageSystemStackTrace>::read(std::__1::vector<std::__1::basic_string<char>,
std::__1::char_traits<char>, std::__1::allocator<char> >, std::__1::allocator<std::__1::basic_string<char>,
std::__1::char_traits<char>, std::__1::allocator<char> > > const&, DB::SelectQueryInfo const&, DB::Context const&, DB::QueryProcessingStage::Enum, unsigned long, unsigned int)
DB::InterpreterSelectQuery::executeFetchColumns(DB::QueryProcessingStage::Enum, DB::QueryPipeline&,
std::__1::shared_ptr<DB::PrewhereInfo> const&, std::__1::vector<std::__1::basic_string<char>,
std::__1::char_traits<char>, std::__1::allocator<char> >, std::__1::allocator<std::__1::basic_string<char>,
std::__1::char_traits<char>, std::__1::allocator<char> > > const&)
DB::InterpreterSelectQuery::executelImpl(DB::QueryPipeline&, std::__1::shared_ptr<DB::IBlockInputStream> const&,
std::__1::optional<DB::Pipe>)
DB::InterpreterSelectQuery::execute()
DB::InterpreterSelectQueryWithUnionQuery::execute()
DB::executeQueryImpl(char const*, char const*, DB::Context&, bool, DB::QueryProcessingStage::Enum, bool,
DB::ReadBuffer*)
DB::executeQuery(std::__1::basic_string<char>, std::__1::char_traits<char>, std::__1::allocator<char> > const&,
DB::Context&, bool, DB::QueryProcessingStage::Enum, bool)
DB::TCPHandler::runImpl()
DB::TCPHandler::run()
Poco::Net::TCPServerConnection::start()
Poco::Net::TCPServerDispatcher::run()
Poco::PooledThread::run()
Poco::ThreadImpl::runnableEntry(void*)
start_thread
__clone
```

Getting filenames and line numbers in ClickHouse source code:

```
WITH arrayMap(x -> addressToLine(x), trace) AS all, arrayFilter(x -> x LIKE '%/dbms/%', all) AS dbms SELECT
thread_name, thread_id, query_id, arrayStringConcat(notEmpty(dbms) ? dbms : all, '\n') AS res FROM
system.stack_trace LIMIT 1 \G;
```

Row 1:

```
thread_name: clickhouse-serv

thread_id: 686
query_id: cad353e7-1c29-4b2e-949f-93e597ab7a54
res: /lib/x86_64-linux-gnu/libc-2.27.so
/build/obj-x86_64-linux-gnu/../src/Storages/System/StorageSystemStackTrace.cpp:182
/build/obj-x86_64-linux-gnu/../contrib/libcxx/include/vector:656
/build/obj-x86_64-linux-gnu/../src/Interpreters/InterpreterSelectQuery.cpp:1338
/build/obj-x86_64-linux-gnu/../src/Interpreters/InterpreterSelectQuery.cpp:751
/build/obj-x86_64-linux-gnu/../contrib/libcxx/include/optional:224
/build/obj-x86_64-linux-gnu/../src/Interpreters/InterpreterSelectWithUnionQuery.cpp:192
/build/obj-x86_64-linux-gnu/../src/Interpreters/executeQuery.cpp:384
/build/obj-x86_64-linux-gnu/../src/Interpreters/executeQuery.cpp:643
/build/obj-x86_64-linux-gnu/../src/Server/TCPHandler.cpp:251
/build/obj-x86_64-linux-gnu/../src/Server/TCPHandler.cpp:1197
/build/obj-x86_64-linux-gnu/../contrib/poco/Net/src/TCPServerConnection.cpp:57
/build/obj-x86_64-linux-gnu/../contrib/libcxx/include/atomic:856
/build/obj-x86_64-linux-gnu/../contrib/poco/Foundation/include/Poco/Mutex_POSIX.h:59
/build/obj-x86_64-linux-gnu/../contrib/poco/Foundation/include/Poco/AutoPtr.h:223
/lib/x86_64-linux-gnu/libpthread-2.27.so
/lib/x86_64-linux-gnu/libc-2.27.so
```

## See Also

- [Introspection Functions](#) — Which introspection functions are available and how to use them.
- [system.trace\\_log](#) — Contains stack traces collected by the sampling query profiler.
- [arrayMap](#) — Description and usage example of the `arrayMap` function.
- [arrayFilter](#) — Description and usage example of the `arrayFilter` function.

## system.storage\_policies

Contains information about storage policies and volumes defined in the [server configuration](#).

Columns:

- `policy_name` ([String](#)) — Name of the storage policy.
- `volume_name` ([String](#)) — Volume name defined in the storage policy.
- `volume_priority` ([UInt64](#)) — Volume order number in the configuration.
- `disks` ([Array\(String\)](#)) — 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.
- `prefer_not_to_merge` ([UInt8](#)) — Value of the `prefer_not_to_merge` setting. When this setting is enabled, merging data on this volume is not allowed. This allows controlling how ClickHouse works with slow disks.

If the storage policy contains more than one volume, then information for each volume is stored in the individual row of the table.

## system.table\_engines

Contains description of table engines supported by server and their feature support information.

This table contains the following columns (the column type is shown in brackets):

- `name` (String) — The name of table engine.
- `supports_settings` (UInt8) — Flag that indicates if table engine supports `SETTINGS` clause.
- `supports_skipping_indices` (UInt8) — Flag that indicates if table engine supports **skipping indices**.
- `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` and `SAMPLE_BY`.
- `supports_replication` (UInt8) — Flag that indicates if table engine supports **data replication**.
- `supports_deduplication` (UInt8) — Flag that indicates if table engine supports data deduplication.
- `supports_parallel_insert` (UInt8) — Flag that indicates if table engine supports parallel insert (see `max_insert_threads` setting).

Example:

```
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	supports_parallel_insert
MergeTree	1	1	0	1	1	1	0
Kafka	0	1	0	0	0	0	0
ReplicatedCollapsingMergeTree	1	1	1	1	0	1	1

## See also

- MergeTree family **query clauses**
- Kafka **settings**
- Join **settings**

## system.tables

Contains metadata of each table that the server knows about.

**Detached tables** are not shown in `system.tables`.

**Temporary tables** are visible in the `system.tables` only in those session where they have been created. They are shown with the empty `database` field and with the `is_temporary` flag switched on.

Columns:

- `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`) - Flag that indicates whether the table is temporary.
- `data_path` (`String`) - Path to the table data in the file system.
- `metadata_path` (`String`) - Path to the table metadata in the file system.
- `metadata_modification_time` (`DateTime`) - Time of latest modification of the table metadata.
- `dependencies_database` (`Array(String)`) - Database dependencies.
- `dependencies_table` (`Array(String)`) - Table dependencies (`MaterializedView` tables based on the current table).
- `create_table_query` (`String`) - The query that was used to create the table.
- `engine_full` (`String`) - Parameters of the table engine.
- `partition_key` (`String`) - The partition key expression specified in the table.
- `sorting_key` (`String`) - The sorting key expression specified in the table.
- `primary_key` (`String`) - The primary key expression specified in the table.
- `sampling_key` (`String`) - The sampling key expression specified in the table.
- `storage_policy` (`String`) - The storage policy:
  - `MergeTree`
  - `Distributed`
- `total_rows` (`Nullable(UInt64)`) - Total number of rows, if it is possible to quickly determine exact number of rows in the table, otherwise `NULL` (including underlying `Buffer` table).
- `total_bytes` (`Nullable(UInt64)`) - Total number of bytes, if it is possible to quickly determine exact number of bytes for the table on storage, otherwise `NULL` (does not includes any underlying storage).
  - If the table stores data on disk, returns used space on disk (i.e. compressed).
  - If the table stores data in memory, returns approximated number of used bytes in memory.
- `lifetime_rows` (`Nullable(UInt64)`) - Total number of rows INSERTed since server start (only for `Buffer` tables).
- `lifetime_bytes` (`Nullable(UInt64)`) - Total number of bytes INSERTed since server start (only for `Buffer` tables).
- `comment` (`String`) - The comment for the table.

The `system.tables` table is used in `SHOW TABLES` query implementation.

## Example

```
SELECT * FROM system.tables LIMIT 2 FORMAT Vertical;
```

Row 1:

```
database:      base
name:          t1
uuid:          81b1c20a-b7c6-4116-a2ce-7583fb6b6736
engine:        MergeTree
is_temporary:  0
data_paths:    ['/var/lib/clickhouse/store/81b/81b1c20a-b7c6-4116-a2ce-7583fb6b6736/']
metadata_path: /var/lib/clickhouse/store/461/461cf698-fd0b-406d-8c01-5d8fd5748a91/t1.sql
metadata_modification_time: 2021-01-25 19:14:32
dependencies_database: []
dependencies_table: []
create_table_query: CREATE TABLE base.t1 (`n` UInt64) ENGINE = MergeTree ORDER BY n SETTINGS
index_granularity = 8192
engine_full:   MergeTree ORDER BY n SETTINGS index_granularity = 8192
partition_key:
sorting_key:   n
primary_key:   n
sampling_key:
storage_policy: default
total_rows:    1
total_bytes:   99
lifetime_rows: NULL
lifetime_bytes: NULL
comment:
```

Row 2:

```
database:      default
name:          53r93yleapyears
uuid:          00000000-0000-0000-0000-000000000000
engine:        MergeTree
is_temporary:  0
data_paths:    ['/var/lib/clickhouse/data/default/53r93yleapyears/']
metadata_path: /var/lib/clickhouse/metadata/default/53r93yleapyears.sql
metadata_modification_time: 2020-09-23 09:05:36
dependencies_database: []
dependencies_table: []
create_table_query: CREATE TABLE default.`53r93yleapyears`(`id` Int8, `febdays` Int8) ENGINE = MergeTree
ORDER BY id SETTINGS index_granularity = 8192
engine_full:   MergeTree ORDER BY id SETTINGS index_granularity = 8192
partition_key:
sorting_key:   id
primary_key:   id
sampling_key:
storage_policy: default
total_rows:    2
total_bytes:   155
lifetime_rows: NULL
lifetime_bytes: NULL
comment:
```

## system.text\_log

Contains logging entries. The logging level which goes to this table can be limited to the `text_log.level` server setting.

Columns:

- `event_date` (Date) — Date of the entry.
- `event_time` (DateTime) — Time of the entry.
- `event_time_microseconds` (DateTime) — Time of the entry with microseconds precision.
- `microseconds` (UInt32) — Microseconds of the entry.
- `thread_name` (String) — Name of the thread from which the logging was done.

- `thread_id` (UInt64) — OS thread ID.
- `level` (Enum8) — Entry level. Possible values:
  - 1 or 'Fatal'.
  - 2 or 'Critical'.
  - 3 or 'Error'.
  - 4 or 'Warning'.
  - 5 or 'Notice'.
  - 6 or 'Information'.
  - 7 or 'Debug'.
  - 8 or 'Trace'.
- `query_id` (String) — ID of the query.
- `logger_name` (LowCardinality(String)) — Name of the logger (i.e. `DDLWorker`).
- `message` (String) — The message itself.
- `revision` (UInt32) — ClickHouse revision.
- `source_file` (LowCardinality(String)) — Source file from which the logging was done.
- `source_line` (UInt64) — Source line from which the logging was done.

## Example

```
SELECT * FROM system.text_log LIMIT 1 \G
```

Row 1:

<code>event_date:</code>	2020-09-10
<code>event_time:</code>	2020-09-10 11:23:07
<code>event_time_microseconds:</code>	2020-09-10 11:23:07.871397
<code>microseconds:</code>	871397
<code>thread_name:</code>	clickhouse-serv
<code>thread_id:</code>	564917
<code>level:</code>	Information
<code>query_id:</code>	
<code>logger_name:</code>	DNSCacheUpdater
<code>message:</code>	Update period 15 seconds
<code>revision:</code>	54440
<code>source_file:</code>	/ClickHouse/src/Interpreters/DNSCacheUpdater.cpp; void DB::DNSCacheUpdater::start()
<code>source_line:</code>	45

## system.time\_zones

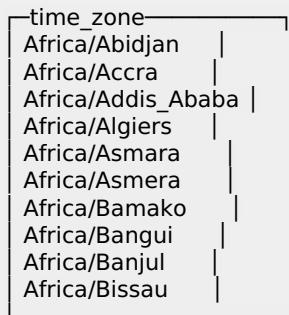
Contains a list of time zones that are supported by the ClickHouse server. This list of timezones might vary depending on the version of ClickHouse.

Columns:

- `time_zone` (String) — List of supported time zones.

## Example

```
SELECT * FROM system.time_zones LIMIT 10
```



## system.trace\_log

Contains stack traces collected by the sampling query profiler.

ClickHouse creates this table when the `trace_log` server configuration section is set. Also the `query_profiler_real_time_period_ns` and `query_profiler_cpu_time_period_ns` settings should be set.

To analyze logs, use the `addressToLine`, `addressToSymbol` and `demangle` introspection functions.

Columns:

- `event_date` (`Date`) — Date of sampling moment.
- `event_time` (`DateTime`) — Timestamp of the sampling moment.
- `event_time_microseconds` (`DateTime64`) — Timestamp of the sampling moment with microseconds precision.
- `timestamp_ns` (`UInt64`) — Timestamp of the sampling moment in nanoseconds.
- `revision` (`UInt32`) — ClickHouse server build revision.

When connecting to the server by `clickhouse-client`, you see the string similar to `Connected to ClickHouse server version 19.18.1 revision 54429..`. This field contains the `revision`, but not the version of a server.

- `trace_type` (`Enum8`) — Trace type:
  - `Real` represents collecting stack traces by wall-clock time.
  - `CPU` represents collecting stack traces by CPU time.
  - `Memory` represents collecting allocations and deallocations when memory allocation exceeds the subsequent watermark.
  - `MemorySample` represents collecting random allocations and deallocations.
- `thread_number` (`UInt32`) — Thread identifier.
- `query_id` (`String`) — Query identifier that can be used to get details about a query that was running from the `query_log` system table.
- `trace` (`Array(UInt64)`) — Stack trace at the moment of sampling. Each element is a virtual memory address inside ClickHouse server process.

### Example

```
SELECT * FROM system.trace_log LIMIT 1 \G
```

Row 1:

```
event_date: 2020-09-10
event_time: 2020-09-10 11:23:09
event_time_microseconds: 2020-09-10 11:23:09.872924
timestamp_ns: 1599762189872924510
revision: 54440
trace_type: Memory
thread_id: 564963
query_id:
trace:
[371912858,371912789,371798468,371799717,371801313,371790250,624462773,566365041,566440261,5664458
34,566460071,566459914,566459842,566459580,566459469,566459389,566459341,566455774,371993941,37198
8245,372158848,372187428,372187309,372187093,372185478,140222123165193,140222122205443]
size: 5244400
```

## system.users

Contains a list of **user accounts** configured at the server.

Columns:

- `name` (**String**) — User name.
- `id` (**UUID**) — User ID.
- `storage` (**String**) — Path to the storage of users. Configured in the `access_control_path` parameter.
- `auth_type` (**Enum8**('no\_password' = 0,'plaintext\_password' = 1, 'sha256\_password' = 2, 'double\_sha1\_password' = 3)) — Shows the authentication type. There are multiple ways of user identification: with no password, with plain text password, with **SHA256**-encoded password or with **double SHA-1**-encoded password.
- `auth_params` (**String**) — Authentication parameters in the JSON format depending on the `auth_type`.
- `host_ip` (**Array(String)**) — IP addresses of hosts that are allowed to connect to the ClickHouse server.
- `host_names` (**Array(String)**) — Names of hosts that are allowed to connect to the ClickHouse server.
- `host_names_regexp` (**Array(String)**) — Regular expression for host names that are allowed to connect to the ClickHouse server.
- `host_names_like` (**Array(String)**) — Names of hosts that are allowed to connect to the ClickHouse server, set using the LIKE predicate.
- `default_roles_all` (**UInt8**) — Shows that all granted roles set for user by default.
- `default_roles_list` (**Array(String)**) — List of granted roles provided by default.
- `default_roles_except` (**Array(String)**) — All the granted roles set as default excepting of the listed ones.

## See Also

- [SHOW USERS](#)

## system.zookeeper

The table does not exist if ZooKeeper is not configured. Allows reading data from the ZooKeeper cluster defined in the config.

The query must either have a 'path =' condition or a path IN condition set with the WHERE clause as shown below. This corresponds to the path of the children in ZooKeeper that you want to get data for.

The query `SELECT * FROM system.zookeeper WHERE path = '/clickhouse'` outputs data for all children on the `/clickhouse` node.

To output data for all root nodes, write `path = '/'`.

If the path specified in 'path' does not exist, an exception will be thrown.

The query `SELECT * FROM system.zookeeper WHERE path IN ('/', '/clickhouse')` outputs data for all children on the `/` and `/clickhouse` node.

If in the specified 'path' collection has does not exist path, an exception will be thrown.

It can be used to do a batch of ZooKeeper path queries.

Columns:

- `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.

Example:

```
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
mzxid: 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
mzxid: 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
```

## system.zookeeper\_log

This table contains information about the parameters of the request to the ZooKeeper server and the response from it.

For requests, only columns with request parameters are filled in, and the remaining columns are filled with default values (0 or `NULL`). When the response arrives, the data from the response is added to the other columns.

Columns with request parameters:

- `type` (**Enum**) — Event type in the ZooKeeper client. Can have one of the following values:
  - `Request` — The request has been sent.
  - `Response` — The response was received.
  - `Finalize` — The connection is lost, no response was received.
- `event_date` (**Date**) — The date when the event happened.
- `event_time` (**DateTime64**) — The date and time when the event happened.
- `address` (**IPv6**) — IP address of ZooKeeper server that was used to make the request.
- `port` (**UInt16**) — The port of ZooKeeper server that was used to make the request.
- `session_id` (**Int64**) — The session ID that the ZooKeeper server sets for each connection.
- `xid` (**Int32**) — The ID of the request within the session. This is usually a sequential request number. It is the same for the request row and the paired `response/finalize` row.
- `has_watch` (**UInt8**) — The request whether the `watch` has been set.

- `op_num` (**Enum**) — The type of request or response.
- `path` (**String**) — The path to the ZooKeeper node specified in the request, or an empty string if the request does not require specifying a path.
- `data` (**String**) — The data written to the ZooKeeper node (for the `SET` and `CREATE` requests — what the request wanted to write, for the response to the `GET` request — what was read) or an empty string.
- `is_ephemeral` (**UInt8**) — Is the ZooKeeper node being created as an **ephemeral**.
- `is_sequential` (**UInt8**) — Is the ZooKeeper node being created as an **sequential**.
- `version` (**Nullable(Int32)**) — The version of the ZooKeeper node that the request expects when executing. This is supported for `CHECK`, `SET`, `REMOVE` requests (is relevant `-1` if the request does not check the version or `NULL` for other requests that do not support version checking).
- `requests_size` (**UInt32**) — The number of requests included in the multi request (this is a special request that consists of several consecutive ordinary requests and executes them atomically). All requests included in multi request will have the same `xid`.
- `request_idx` (**UInt32**) — The number of the request included in multi request (for multi request — `0`, then in order from `1`).

Columns with request response parameters:

- `xzid` (**Int64**) — ZooKeeper transaction ID. The serial number issued by the ZooKeeper server in response to a successfully executed request (`0` if the request was not executed/returned an error/the client does not know whether the request was executed).
- `error` (**Nullable(Enum)**) — Error code. Can have many values, here are just some of them:
  - `ZOK` — The request was executed successfully.
  - `ZCONNECTIONLOSS` — The connection was lost.
  - `ZOPERATIONTIMEOUT` — The request execution timeout has expired.
  - `ZSESSIONEXPIRED` — The session has expired.
  - `NULL` — The request is completed.
- `watch_type` (**Nullable(Enum)**) — The type of the `watch` event (for responses with `op_num = Watch`), for the remaining responses: `NULL`.
- `watch_state` (**Nullable(Enum)**) — The status of the `watch` event (for responses with `op_num = Watch`), for the remaining responses: `NULL`.
- `path_created` (**String**) — The path to the created ZooKeeper node (for responses to the `CREATE` request), may differ from the `path` if the node is created as a `sequential`.
- `stat_czxid` (**Int64**) — The `xzid` of the change that caused this ZooKeeper node to be created.
- `stat_mzxid` (**Int64**) — The `xzid` of the change that last modified this ZooKeeper node.
- `stat_pzxid` (**Int64**) — The transaction ID of the change that last modified children of this ZooKeeper node.
- `stat_version` (**Int32**) — The number of changes to the data of this ZooKeeper node.
- `stat_cversion` (**Int32**) — The number of changes to the children of this ZooKeeper node.
- `stat_dataLength` (**Int32**) — The length of the data field of this ZooKeeper node.
- `stat_numChildren` (**Int32**) — The number of children of this ZooKeeper node.

- `children` ([Array\(String\)](#)) — The list of child ZooKeeper nodes (for responses to LIST request).

## Example

Query:

```
SELECT * FROM system.zookeeper_log WHERE (session_id = '106662742089334927') AND (xid = '10858') FORMAT  
Vertical;
```

Result:

Row 1:

```
type: Request
event_date: 2021-08-09
event_time: 2021-08-09 21:38:30.291792
address: ::
port: 2181
session_id: 106662742089334927
xid: 10858
has_watch: 1
op_num: List
path: /clickhouse/task_queue/ddl
data:
is_ephemeral: 0
is_sequential: 0
version: NULL
requests_size: 0
request_idx: 0
zxid: 0
error: NULL
watch_type: NULL
watch_state: NULL
path_created:
stat_czid: 0
stat_mzid: 0
stat_pzid: 0
stat_version: 0
stat_cversion: 0
stat_dataLength: 0
stat_numChildren: 0
children: []
```

Row 2:

```
type: Response
event_date: 2021-08-09
event_time: 2021-08-09 21:38:30.292086
address: ::
port: 2181
session_id: 106662742089334927
xid: 10858
has_watch: 1
op_num: List
path: /clickhouse/task_queue/ddl
data:
is_ephemeral: 0
is_sequential: 0
version: NULL
requests_size: 0
request_idx: 0
zxid: 16926267
error: ZOK
watch_type: NULL
watch_state: NULL
path_created:
stat_czid: 16925469
stat_mzid: 16925469
stat_pzid: 16926179
stat_version: 0
stat_cversion: 7
stat_dataLength: 0
stat_numChildren: 7
children: ['query-0000000006','query-0000000005','query-0000000004','query-0000000003','query-0000000002','query-0000000001','query-0000000000']
```

## See Also

- [ZooKeeper](#)
- [ZooKeeper guide](#)

# Server Configuration Parameters

This section contains descriptions of server settings that cannot be changed at the session or query level.

These settings are stored in the `config.xml` file on the ClickHouse server.

Other settings are described in the “[Settings](#)” section.

Before studying the settings, read the [Configuration files](#) section and note the use of substitutions (the `incl` and `optional` attributes).

## Server Settings

### `builtin_dictionaries_reload_interval`

The interval in seconds before reloading built-in dictionaries.

ClickHouse reloads built-in dictionaries every x seconds. This makes it possible to edit dictionaries “on the fly” without restarting the server.

Default value: 3600.

#### Example

```
<builtin_dictionaries_reload_interval>3600</builtin_dictionaries_reload_interval>
```

### `compression`

Data compression settings for [MergeTree](#)-engine tables.

#### Warning

Don’t use it if you have just started using ClickHouse.

Configuration template:

```
<compression>
  <case>
    <min_part_size>...</min_part_size>
    <min_part_size_ratio>...</min_part_size_ratio>
    <method>...</method>
    <level>...</level>
  </case>
  ...
</compression>
```

`<case>` fields:

- `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`, `lz4hc`, `zstd`.
- `level` – Compression level. See [Codecs](#).

You can configure multiple `<case>` sections.

Actions when conditions are met:

- If a data part matches a condition set, ClickHouse uses the specified compression method.
- If a data part matches multiple condition sets, ClickHouse uses the first matched condition set.

If no conditions met for a data part, ClickHouse uses the `lz4` compression.

### Example

```
<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>
    <level>1</level>
  </case>
</compression>
```

## custom\_settings\_prefixes

List of prefixes for [custom settings](#). The prefixes must be separated with commas.

### Example

```
<custom_settings_prefixes>custom_</custom_settings_prefixes>
```

### See Also

- [Custom settings](#)

## core\_dump

Configures soft limit for core dump file size.

Possible values:

- Positive integer.

Default value: `1073741824` (1 GB).

### Note

Hard limit is configured via system tools

### Example

```
<core_dump>
  <size_limit>1073741824</size_limit>
</core_dump>
```

## database\_atomic\_delay\_before\_drop\_table\_sec

Sets the delay before remove table data in seconds. If the query has `SYNC` modifier, this setting is ignored.

Default value: `480` (8 minute).

## default\_database

The default database.

To get a list of databases, use the **SHOW DATABASES** query.

### Example

```
<default_database>default</default_database>
```

## default\_profile

Default settings profile.

Settings profiles are located in the file specified in the parameter `user_config`.

### Example

```
<default_profile>default</default_profile>
```

## default\_replica\_path

The path to the table in ZooKeeper.

### Example

```
<default_replica_path>/clickhouse/tables/{uuid}/{shard}</default_replica_path>
```

## default\_replica\_name

The replica name in ZooKeeper.

### Example

```
<default_replica_name>{replica}</default_replica_name>
```

## dictionaries\_config

The path to the config file for external dictionaries.

Path:

- Specify the absolute path or the path relative to the server config file.
- The path can contain wildcards \* and ?.

See also “[External dictionaries](#)”.

### Example

```
<dictionaries_config>*_dictionary.xml</dictionaries_config>
```

## dictionaries\_lazy\_load

Lazy loading of dictionaries.

If `true`, then each dictionary is created on first use. If dictionary creation failed, the function that was using the dictionary throws an exception.

If false, all dictionaries are created when the server starts, if the dictionary or dictionaries are created too long or are created with errors, then the server boots without of these dictionaries and continues to try to create these dictionaries.

The default is true.

### Example

```
<dictionaries_lazy_load>true</dictionaries_lazy_load>
```

## format\_schema\_path

The path to the directory with the schemes for the input data, such as schemas for the [CapnProto](#) format.

### Example

```
<!-- Directory containing schema files for various input formats. -->
<format_schema_path>format_schemas/</format_schema_path>
```

## graphite

Sending data to [Graphite](#).

Settings:

- 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 [system.metrics](#) table.
- events - Sending deltas data accumulated for the time period from the [system.events](#) table.
- events\_cumulative - Sending cumulative data from the [system.events](#) table.
- asynchronous\_metrics - Sending data from the [system.asynchronous\\_metrics](#) table.

You can configure multiple `<graphite>` clauses. For instance, you can use this for sending different data at different intervals.

### Example

```
<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

Settings for thinning data for Graphite.

For more details, see [GraphiteMergeTree](#).

### Example

```
<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

The port for connecting to the server over HTTP(s).

If `https_port` is specified, [openSSL](#) must be configured.

If `http_port` is specified, the OpenSSL configuration is ignored even if it is set.

### Example

```
<https_port>9999</https_port>
```

## http\_server\_default\_response

The page that is shown by default when you access the ClickHouse HTTP(s) server.

The default value is “Ok.” (with a line feed at the end)

### Example

Opens <https://tabix.io/> when accessing `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="" class="content-ui"></div><script src="http://loader.tabix.io/master.js"></script></body></html>]]>
</http_server_default_response>
```

## hsts\_max\_age

Expired time for HSTS in seconds. The default value is 0 means clickhouse disabled HSTS. If you set a positive number, the HSTS will be enabled and the max-age is the number you set.

### Example

```
<hsts_max_age>600000</hsts_max_age>
```

## include\_from

The path to the file with substitutions.

For more information, see the section “[Configuration files](#)”.

### **Example**

```
<include_from>/etc/metrica.xml</include_from>
```

## interserver\_http\_port

Port for exchanging data between ClickHouse servers.

### **Example**

```
<interserver_http_port>9009</interserver_http_port>
```

## interserver\_http\_host

The hostname that can be used by other servers to access this server.

If omitted, it is defined in the same way as the `hostname-f` command.

Useful for breaking away from a specific network interface.

### **Example**

```
<interserver_http_host>example.yandex.ru</interserver_http_host>
```

## interserver\_https\_port

Port for exchanging data between ClickHouse servers over [HTTPS](#).

### **Example**

```
<interserver_https_port>9010</interserver_https_port>
```

## interserver\_https\_host

Similar to `interserver_http_host`, except that this hostname can be used by other servers to access this server over [HTTPS](#).

### **Example**

```
<interserver_https_host>example.yandex.ru</interserver_https_host>
```

## interserver\_http\_credentials

The username and password used to authenticate during [replication](#) with the Replicated\* engines. These credentials are used only for communication between replicas and are unrelated to credentials for ClickHouse clients. The server is checking these credentials for connecting replicas and use the same credentials when connecting to other replicas. So, these credentials should be set the same for all replicas in a cluster.

By default, the authentication is not used.

## Note

These credentials are common for replication through [HTTP](#) and [HTTPS](#).

This section contains the following parameters:

- `user` — username.
- `password` — password.

### Example

```
<interserver_http_credentials>
  <user>admin</user>
  <password>222</password>
</interserver_http_credentials>
```

## keep\_alive\_timeout

The number of seconds that ClickHouse waits for incoming requests before closing the connection.  
Defaults to 10 seconds.

### Example

```
<keep_alive_timeout>10</keep_alive_timeout>
```

## listen\_host

Restriction on hosts that requests can come from. If you want the server to answer all of them, specify `::`.

Examples:

```
<listen_host>::1</listen_host>
<listen_host>127.0.0.1</listen_host>
```

## listen\_backlog

Backlog (queue size of pending connections) of the listen socket.

Default value: `4096` (as in linux [5.4+](#)).

Usually this value does not need to be changed, since:

- default value is large enough,
- and for accepting client's connections server has separate thread.

So even if you have `TcpExtListenOverflows` (from `nstat`) non zero and this counter grows for ClickHouse server it does not mean that this value need to be increased, since:

- usually if 4096 is not enough it shows some internal ClickHouse scaling issue, so it is better to report an issue.
- and it does not mean that the server can handle more connections later (and even if it can, clients can already goes away / disconnect).

Examples:

```
<listen_backlog>4096</listen_backlog>
```

## logger

Logging settings.

Keys:

- `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` and `errorlog`. Once the file reaches `size`, ClickHouse archives and renames it, and creates a new log file in its place.
- `count` – The number of archived log files that ClickHouse stores.

### Example

```
<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>
```

Writing to the syslog is also supported. Config example:

```
<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>
```

Keys for syslog:

- `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` — **The syslog facility keyword** in uppercase letters with the “LOG\_” prefix: (`LOG_USER`, `LOG_DAEMON`, `LOG_LOCAL3`, and so on).  
Default value: `LOG_USER` if `address` is specified, `LOG_DAEMON` otherwise.
- `format` – Message format. Possible values: `bsd` and `syslog`.

## send\_crash\_reports

Settings for opt-in sending crash reports to the ClickHouse core developers team via [Sentry](#). Enabling it, especially in pre-production environments, is highly appreciated.

The server will need access to the public Internet via IPv4 (at the time of writing IPv6 is not supported by Sentry) for this feature to be functioning properly.

Keys:

- `enabled` – Boolean flag to enable the feature, `false` by default. Set to `true` to allow sending crash reports.
- `endpoint` – You can override the Sentry endpoint URL for sending crash reports. It can be either a separate Sentry account or your self-hosted Sentry instance. Use the [Sentry DSN](#) syntax.
- `anonymize` - Avoid attaching the server hostname to the crash report.
- `http_proxy` - Configure HTTP proxy for sending crash reports.
- `debug` - Sets the Sentry client into debug mode.
- `tmp_path` - Filesystem path for temporary crash report state.

### **Recommended way to use**

```
<send_crash_reports>
  <enabled>true</enabled>
</send_crash_reports>
```

## macros

Parameter substitutions for replicated tables.

Can be omitted if replicated tables are not used.

For more information, see the section [Creating replicated tables](#).

### **Example**

```
<macros incl="macros" optional="true" />
```

## mark\_cache\_size

Approximate size (in bytes) of the cache of marks used by table engines of the [MergeTree](#) family.

The cache is shared for the server and memory is allocated as needed. The cache size must be at least 5368709120.

### **Example**

```
<mark_cache_size>5368709120</mark_cache_size>
```

## max\_server\_memory\_usage

Limits total RAM usage by the ClickHouse server.

Possible values:

- Positive integer.
- 0 — Auto.

Default value: 0.

## Additional Info

The default `max_server_memory_usage` value is calculated as `memory_amount * max_server_memory_usage_to_ram_ratio`.

## See also

- [max\\_memory\\_usage](#)
- [max\\_server\\_memory\\_usage\\_to\\_ram\\_ratio](#)

## max\_server\_memory\_usage\_to\_ram\_ratio

Defines the fraction of total physical RAM amount, available to the ClickHouse server. If the server tries to utilize more, the memory is cut down to the appropriate amount.

Possible values:

- Positive double.
- 0 — The ClickHouse server can use all available RAM.

Default value: 0.9.

## Usage

On hosts with low RAM and swap, you possibly need setting `max_server_memory_usage_to_ram_ratio` larger than 1.

## Example

```
<max_server_memory_usage_to_ram_ratio>0.9</max_server_memory_usage_to_ram_ratio>
```

## See Also

- [max\\_server\\_memory\\_usage](#)

## max\_concurrent\_queries

The maximum number of simultaneously processed queries related to MergeTree table. Queries may be limited by other settings: [max\\_concurrent\\_queries\\_for\\_user](#), [max\\_concurrent\\_queries\\_for\\_all\\_users](#), [min\\_marks\\_to\\_honor\\_max\\_concurrent\\_queries](#).

### Note

These settings can be modified at runtime and will take effect immediately. Queries that are already running will remain unchanged.

Possible values:

- Positive integer.
- 0 — Disabled.

## Example

```
<max_concurrent_queries>100</max_concurrent_queries>
```

## **max\_concurrent\_queries\_for\_user**

The maximum number of simultaneously processed queries related to MergeTree table per user.

Possible values:

- Positive integer.
- 0 — Disabled.

### **Example**

```
<max_concurrent_queries_for_user>5</max_concurrent_queries_for_user>
```

## **max\_concurrent\_queries\_for\_all\_users**

Throw exception if the value of this setting is less or equal than the current number of simultaneously processed queries.

Example: `max_concurrent_queries_for_all_users` can be set to 99 for all users and database administrator can set it to 100 for itself to run queries for investigation even when the server is overloaded.

Modifying the setting for one query or user does not affect other queries.

Default value: 0 that means no limit.

### **Example**

```
<max_concurrent_queries_for_all_users>99</max_concurrent_queries_for_all_users>
```

### **See Also**

- [max\\_concurrent\\_queries](#)

## **min\_marks\_to\_honor\_max\_concurrent\_queries**

The minimal number of marks read by the query for applying the [max\\_concurrent\\_queries](#) setting.

Possible values:

- Positive integer.
- 0 — Disabled.

### **Example**

```
<min_marks_to_honor_max_concurrent_queries>10</min_marks_to_honor_max_concurrent_queries>
```

## **max\_connections**

The maximum number of inbound connections.

### **Example**

```
<max_connections>4096</max_connections>
```

## **max\_open\_files**

The maximum number of open files.

By default: `maximum`.

We recommend using this option in Mac OS X since the `getrlimit()` function returns an incorrect value.

### Example

```
<max_open_files>262144</max_open_files>
```

## max\_table\_size\_to\_drop

Restriction on deleting tables.

If the size of a [MergeTree](#) table exceeds `max_table_size_to_drop` (in bytes), you can't delete it using a `DROP` query.

If you still need to delete the table without restarting the ClickHouse server, create the `<clickhouse-path>/flags/force_drop_table` file and run the `DROP` query.

Default value: 50 GB.

The value 0 means that you can delete all tables without any restrictions.

### Example

```
<max_table_size_to_drop>0</max_table_size_to_drop>
```

## max\_thread\_pool\_size

The maximum number of threads in the Global Thread pool.

Default value: 10000.

### Example

```
<max_thread_pool_size>12000</max_thread_pool_size>
```

## merge\_tree

Fine tuning for tables in the [MergeTree](#).

For more information, see the `MergeTreeSettings.h` header file.

### Example

```
<merge_tree>
  <max_suspicious_broken_parts>5</max_suspicious_broken_parts>
</merge_tree>
```

## metric\_log

It is enabled by default. If it's not, you can do this manually.

### Enabling

To manually turn on metrics history collection `system.metric_log`, create `/etc/clickhouse-server/config.d/metric_log.xml` with the following content:

```
<clickhouse>
  <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>
</clickhouse>
```

## Disabling

To disable `metric_log` setting, you should create the following file `/etc/clickhouse-server/config.d/disable_metric_log.xml` with the following content:

```
<clickhouse>
<metric_log remove="1" />
</clickhouse>
```

## replicated\_merge\_tree

Fine tuning for tables in the [ReplicatedMergeTree](#).

This setting has a higher priority.

For more information, see the `MergeTreeSettings.h` header file.

## Example

```
<replicated_merge_tree>
  <max_suspicious_broken_parts>5</max_suspicious_broken_parts>
</replicated_merge_tree>
```

## openSSL

SSL client/server configuration.

Support for SSL is provided by the `libpoco` library. The interface is described in the file [SSLManager.h](#)

Keys for server/client settings:

- `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` contains the certificate.
- `caConfig` – The path to the file or directory that contains trusted root certificates.
- `verificationMode` – The method for checking the node's certificates. Details are in the description of the [Context](#) class. Possible values: `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`. Acceptable values: `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`. This parameter is always recommended since it helps avoid problems both if the server caches the session and if the client requested caching. Default value:  `${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.

### **Example of settings:**

```

<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

Logging events that are associated with [MergeTree](#). For instance, adding or merging data. You can use the log to simulate merge algorithms and compare their characteristics. You can visualize the merge process.

Queries are logged in the [system.part\\_log](#) table, not in a separate file. You can configure the name of this table in the `table` parameter (see below).

Use the following parameters to configure logging:

- `database` – Name of the database.
- `table` – Name of the system table.
- `partition_by` — [Custom partitioning key](#) for a system table. Can't be used if `engine` defined.
- `engine` - [MergeTree Engine Definition](#) for a system table. Can't be used if `partition_by` defined.
- `flush_interval_milliseconds` – Interval for flushing data from the buffer in memory to the table.

### Example

```
<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>
```

## path

The path to the directory containing data.

### Warning

The trailing slash is mandatory.

### Example

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

## prometheus

Exposing metrics data for scraping from [Prometheus](#).

Settings:

- `endpoint` – HTTP endpoint for scraping metrics by prometheus server. Start from '/'.
- `port` – Port for `endpoint`.
- `metrics` – Flag that sets to expose metrics from the [system.metrics](#) table.
- `events` – Flag that sets to expose metrics from the [system.events](#) table.
- `asynchronous_metrics` – Flag that sets to expose current metrics values from the [system.asynchronous\\_metrics](#) table.

## Example

```
<prometheus>
  <endpoint>/metrics</endpoint>
  <port>8001</port>
  <metrics>true</metrics>
  <events>true</events>
  <asynchronous_metrics>true</asynchronous_metrics>
</prometheus>
```

## query\_log

Setting for logging queries received with the `log_queries=1` setting.

Queries are logged in the `system.query_log` table, not in a separate file. You can change the name of the table in the `table` parameter (see below).

Use the following parameters to configure logging:

- `database` – Name of the database.
- `table` – Name of the system table the queries will be logged in.
- `partition_by` — [Custom partitioning key](#) for a system table. Can't be used if `engine` defined.
- `engine` - [MergeTree Engine Definition](#) for a system table. Can't be used if `partition_by` defined.
- `flush_interval_milliseconds` – Interval for flushing data from the buffer in memory to the table.

If the table does not exist, ClickHouse will create it. If the structure of the query log changed when the ClickHouse server was updated, the table with the old structure is renamed, and a new table is created automatically.

## Example

```
<query_log>
  <database>system</database>
  <table>query_log</table>
  <engine>Engine = MergeTree PARTITION BY event_date ORDER BY event_time TTL event_date + INTERVAL 30 day</engine>
  <flush_interval_milliseconds>7500</flush_interval_milliseconds>
</query_log>
```

## query\_thread\_log

Setting for logging threads of queries received with the `log_query_threads=1` setting.

Queries are logged in the `system.query_thread_log` table, not in a separate file. You can change the name of the table in the `table` parameter (see below).

Use the following parameters to configure logging:

- `database` – Name of the database.
- `table` – Name of the system table the queries will be logged in.
- `partition_by` — [Custom partitioning key](#) for a system table. Can't be used if `engine` defined.
- `engine` - [MergeTree Engine Definition](#) for a system table. Can't be used if `partition_by` defined.
- `flush_interval_milliseconds` – Interval for flushing data from the buffer in memory to the table.

If the table does not exist, ClickHouse will create it. If the structure of the query thread log changed when the ClickHouse server was updated, the table with the old structure is renamed, and a new table is created automatically.

### Example

```
<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>
```

## query\_views\_log

Setting for logging views (live, materialized etc) dependant of queries received with the [log\\_query\\_views=1](#) setting.

Queries are logged in the [system.query\\_views\\_log](#) table, not in a separate file. You can change the name of the table in the `table` parameter (see below).

Use the following parameters to configure logging:

- `database` – Name of the database.
- `table` – Name of the system table the queries will be logged in.
- `partition_by` — [Custom partitioning key](#) for a system table. Can't be used if `engine` defined.
- `engine` - [MergeTree Engine Definition](#) for a system table. Can't be used if `partition_by` defined.
- `flush_interval_milliseconds` – Interval for flushing data from the buffer in memory to the table.

If the table does not exist, ClickHouse will create it. If the structure of the query views log changed when the ClickHouse server was updated, the table with the old structure is renamed, and a new table is created automatically.

### Example

```
<query_views_log>
  <database>system</database>
  <table>query_views_log</table>
  <partition_by>toYYYYMM(event_date)</partition_by>
  <flush_interval_milliseconds>7500</flush_interval_milliseconds>
</query_views_log>
```

## text\_log

Settings for the [text\\_log](#) system table for logging text messages.

Parameters:

- `level` — Maximum Message Level (by default `Trace`) which will be stored in a table.
- `database` — Database name.
- `table` — Table name.
- `partition_by` — [Custom partitioning key](#) for a system table. Can't be used if `engine` defined.
- `engine` - [MergeTree Engine Definition](#) for a system table. Can't be used if `partition_by` defined.

- `flush_interval_milliseconds` — Interval for flushing data from the buffer in memory to the table.

## Example

```
<clickhouse>
  <text_log>
    <level>notice</level>
    <database>system</database>
    <table>text_log</table>
    <flush_interval_milliseconds>7500</flush_interval_milliseconds>
    <!-- <partition_by>event_date</partition_by> -->
    <engine>Engine = MergeTree PARTITION BY event_date ORDER BY event_time TTL event_date + INTERVAL 30
day</engine>
  </text_log>
</clickhouse>
```

## trace\_log

Settings for the [trace\\_log](#) system table operation.

Parameters:

- `database` — Database for storing a table.
- `table` — Table name.
- `partition_by` — [Custom partitioning key](#) for a system table. Can't be used if `engine` defined.
- `engine` - [MergeTree Engine Definition](#) for a system table. Can't be used if `partition_by` defined.
- `flush_interval_milliseconds` — Interval for flushing data from the buffer in memory to the table.

The default server configuration file `config.xml` contains the following settings section:

```
<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

Regexp-based rules, which will be applied to queries as well as all log messages before storing them in server logs,

`system.query_log`, `system.text_log`, `system.processes` tables, and in logs sent to the client. That allows preventing sensitive data leakage from SQL queries (like names, emails, personal identifiers or credit card numbers) to logs.

## Example

```
<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>
```

Config fields:

- `name` - name for the rule (optional)
- `regexp` - RE2 compatible regular expression (mandatory)
- `replace` - substitution string for sensitive data (optional, by default - six asterisks)

The masking rules are applied to the whole query (to prevent leaks of sensitive data from malformed / non-parsable queries).

`system.events` table have counter `QueryMaskingRulesMatch` which have an overall number of query masking rules matches.

For distributed queries each server have to be configured separately, otherwise, subqueries passed to other nodes will be stored without masking.

## remote\_servers

Configuration of clusters used by the [Distributed](#) table engine and by the `cluster` table function.

### Example

```
<remote_servers incl="clickhouse_remote_servers" />
```

For the value of the `incl` attribute, see the section “[Configuration files](#)”.

### See Also

- [skip\\_unavailable\\_shards](#)

## timezone

The server’s time zone.

Specified as an IANA identifier for the UTC timezone or geographic location (for example, Africa/Abidjan).

The time zone is necessary for conversions between String and DateTime formats when DateTime fields are output to text format (printed on the screen or in a file), and when getting DateTime from a string. Besides, the time zone is used in functions that work with the time and date if they didn’t receive the time zone in the input parameters.

### Example

```
<timezone>Europe/Moscow</timezone>
```

## tcp\_port

Port for communicating with clients over the TCP protocol.

### Example

```
<tcp_port>9000</tcp_port>
```

## tcp\_port\_secure

TCP port for secure communication with clients. Use it with [OpenSSL](#) settings.

### Possible values

Positive integer.

#### Default value

```
<tcp_port_secure>9440</tcp_port_secure>
```

## mysql\_port

Port for communicating with clients over MySQL protocol.

#### Possible values

Positive integer.

#### Example

```
<mysql_port>9004</mysql_port>
```

## tmp\_path

Path to temporary data for processing large queries.

### Note

The trailing slash is mandatory.

#### Example

```
<tmp_path>/var/lib/clickhouse/tmp/</tmp_path>
```

## tmp\_policy

Policy from [storage\\_configuration](#) to store temporary files.

If not set, [tmp\\_path](#) is used, otherwise it is ignored.

### Note

- [move\\_factor](#) is ignored.
- [keep\\_free\\_space\\_bytes](#) is ignored.
- [max\\_data\\_part\\_size\\_bytes](#) is ignored.
- You must have exactly one volume in that policy.

## uncompressed\_cache\_size

Cache size (in bytes) for uncompressed data used by table engines from the [MergeTree](#).

There is one shared cache for the server. Memory is allocated on demand. The cache is used if the option [use\\_uncompressed\\_cache](#) is enabled.

The uncompressed cache is advantageous for very short queries in individual cases.

## Example

```
<uncompressed_cache_size>8589934592</uncompressed_cache_size>
```

## user\_files\_path

The directory with user files. Used in the table function [file\(\)](#).

## Example

```
<user_files_path>/var/lib/clickhouse/user_files/</user_files_path>
```

## users\_config

Path to the file that contains:

- User configurations.
- Access rights.
- Settings profiles.
- Quota settings.

## Example

```
<users_config>users.xml</users_config>
```

## zookeeper

Contains settings that allow ClickHouse to interact with a [ZooKeeper](#) cluster.

ClickHouse uses ZooKeeper for storing metadata of replicas when using replicated tables. If replicated tables are not used, this section of parameters can be omitted.

This section contains the following parameters:

- `node` — ZooKeeper endpoint. You can set multiple endpoints.

For example:

```
<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_ms` — Maximum timeout for the client session in milliseconds.
- `operation_timeout_ms` — Maximum timeout for one operation in milliseconds.
- `root` — The [znode](#) that is used as the root for znodes used by the ClickHouse server. Optional.
- `identity` — User and password, that can be required by ZooKeeper to give access to requested znodes. Optional.

## Example configuration

```
<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>
```

## See Also

- [Replication](#)
- [ZooKeeper Programmer's Guide](#)

## use\_minimalistic\_part\_header\_in\_zookeeper

Storage method for data part headers in ZooKeeper.

This setting only applies to the `MergeTree` family. It can be specified:

- Globally in the `merge_tree` section of the `config.xml` file.

ClickHouse uses the setting for all the tables on the server. You can change the setting at any time. Existing tables change their behaviour when the setting changes.

- For each table.

When creating a table, specify the corresponding [engine setting](#). The behaviour of an existing table with this setting does not change, even if the global setting changes.

## Possible values

- 0 — Functionality is turned off.
- 1 — Functionality is turned on.

If `use_minimalistic_part_header_in_zookeeper = 1`, then `replicated` tables store the headers of the data parts compactly using a single `znode`. If the table contains many columns, this storage method significantly reduces the volume of the data stored in Zookeeper.

## Attention

After applying `use_minimalistic_part_header_in_zookeeper = 1`, you can't downgrade the ClickHouse server to a version that does not support this setting. Be careful when upgrading ClickHouse on servers in a cluster. Don't upgrade all the servers at once. It is safer to test new versions of ClickHouse in a test environment, or on just a few servers of a cluster.

Data part headers already stored with this setting can't be restored to their previous (non-compact) representation.

**Default value:** 0.

## disable\_internal\_dns\_cache

Disables the internal DNS cache. Recommended for operating ClickHouse in systems with frequently changing infrastructure such as Kubernetes.

**Default value:** 0.

## dns\_cache\_update\_period

The period of updating IP addresses stored in the ClickHouse internal DNS cache (in seconds). The update is performed asynchronously, in a separate system thread.

**Default value:** 15.

### See also

- [background\\_schedule\\_pool\\_size](#)

## distributed\_ddl

Manage executing [distributed ddl queries](#) (CREATE, DROP, ALTER, RENAME) on cluster.

Works only if [ZooKeeper](#) is enabled.

### Example

```
<distributed_ddl>
  <!-- Path in ZooKeeper to queue with DDL queries -->
  <path>/clickhouse/task_queue/ddl</path>

  <!-- Settings from this profile will be used to execute DDL queries -->
  <profile>default</profile>

  <!-- Controls how much ON CLUSTER queries can be run simultaneously. -->
  <pool_size>1</pool_size>

  <!--
    Cleanup settings (active tasks will not be removed)
  -->

  <!-- Controls task TTL (default 1 week) -->
  <task_max_lifetime>604800</task_max_lifetime>

  <!-- Controls how often cleanup should be performed (in seconds) -->
  <cleanup_delay_period>60</cleanup_delay_period>

  <!-- Controls how many tasks could be in the queue -->
  <max_tasks_in_queue>1000</max_tasks_in_queue>
</distributed_ddl>
```

## access\_control\_path

Path to a folder where a ClickHouse server stores user and role configurations created by SQL commands.

Default value: `/var/lib/clickhouse/access/`.

### See also

- [Access Control and Account Management](#)

## user\_directories

Section of the configuration file that contains settings:

- Path to configuration file with predefined users.
- Path to folder where users created by SQL commands are stored.
- ZooKeeper node path where users created by SQL commands are stored and replicated (experimental).

If this section is specified, the path from `users_config` and `access_control_path` won't be used.

The `user_directories` section can contain any number of items, the order of the items means their precedence (the higher the item the higher the precedence).

## Examples

```
<user_directories>
  <users_xml>
    <path>/etc/clickhouse-server/users.xml</path>
  </users_xml>
  <local_directory>
    <path>/var/lib/clickhouse/access/</path>
  </local_directory>
</user_directories>
```

Users, roles, row policies, quotas, and profiles can be also stored in ZooKeeper:

```
<user_directories>
  <users_xml>
    <path>/etc/clickhouse-server/users.xml</path>
  </users_xml>
  <replicated>
    <zookeeper_path>/clickhouse/access/</zookeeper_path>
  </replicated>
</user_directories>
```

You can also define sections `memory` — means storing information only in memory, without writing to disk, and `ldap` — means storing information on an LDAP server.

To add an LDAP server as a remote user directory of users that are not defined locally, define a single `ldap` section with a following parameters:

- `server` — one of LDAP server names defined in `ldap_servers` config section. This parameter is mandatory and cannot be empty.
- `roles` — section with a list of locally defined roles that will be assigned to each user retrieved from the LDAP server. If no roles are specified, user will not be able to perform any actions after authentication. If any of the listed roles is not defined locally at the time of authentication, the authentication attempt will fail as if the provided password was incorrect.

## Example

```
<ldap>
  <server>my_ldap_server</server>
  <roles>
    <my_local_role1 />
    <my_local_role2 />
  </roles>
</ldap>
```

# How to Test Your Hardware with ClickHouse

You can run basic ClickHouse performance test on any server without installation of ClickHouse packages.

## Automated Run

You can run benchmark with a single script.

1. Download the script.

```
wget https://raw.githubusercontent.com/ClickHouse/ClickHouse/master/benchmark/hardware.sh
```

2. Run the script.

```
chmod a+x ./hardware.sh  
./hardware.sh
```

3. Copy the output and send it to [feedback@clickhouse.com](mailto:feedback@clickhouse.com)

All the results are published here: <https://clickhouse.com/benchmark/hardware/>

## Manual Run

Alternatively you can perform benchmark in the following steps.

1. ssh to the server and download the binary with wget:

```
## For amd64:  
wget https://builds.clickhouse.com/master/amd64/clickhouse  
## For aarch64:  
wget https://builds.clickhouse.com/master/aarch64/clickhouse  
## Then do:  
chmod a+x clickhouse
```

2. Download benchmark files:

```
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
```

3. Download test data according to the [Yandex.Metrica dataset](#) instruction (“hits” table containing 100 million rows).

```
wget https://datasets.clickhouse.com/hits/partitions/hits_100m_obfuscated_v1.tar.xz  
tar xvf hits_100m_obfuscated_v1.tar.xz -C .  
mv hits_100m_obfuscated_v1/* .
```

4. Run the server:

```
./clickhouse server
```

5. Check the data: ssh to the server in another terminal

```
./clickhouse client --query "SELECT count() FROM hits_100m_obfuscated"  
100000000
```

6. Run the benchmark:

```
./benchmark-new.sh hits_100m_obfuscated
```

7. Send the numbers and the info about your hardware configuration to [clickhouse-feedback@yandex-team.com](mailto:clickhouse-feedback@yandex-team.com)

All the results are published here: <https://clickhouse.com/benchmark/hardware/>

## Settings

There are multiple ways to make all the settings described in this section of documentation.

Settings are configured in layers, so each subsequent layer redefines the previous settings.

Ways to configure settings, in order of priority:

- Settings in the `users.xml` server configuration file.

Set in the element `<profiles>`.

- Session settings.

Send `SET setting=value` from the ClickHouse console client in interactive mode.

Similarly, you can use ClickHouse sessions in the HTTP protocol. To do this, you need to specify the `session_id` HTTP parameter.

- Query settings.

- When starting the ClickHouse console client in non-interactive mode, set the startup parameter `--setting=value`.
- When using the HTTP API, pass CGI parameters (`URL?setting_1=value&setting_2=value...`).
- Make settings in the `SETTINGS` clause of the `SELECT` query. The setting value is applied only to that query and is reset to default or previous value after the query is executed.

Settings that can only be made in the server config file are not covered in this section.

## Custom Settings

In addition to the common `settings`, users can define custom settings.

A custom setting name must begin with one of predefined prefixes. The list of these prefixes must be declared in the `custom_settings_prefixes` parameter in the server configuration file.

```
<custom_settings_prefixes>custom_</custom_settings_prefixes>
```

To define a custom setting use `SET` command:

```
SET custom_a = 123;
```

To get the current value of a custom setting use `getSetting()` function:

```
SELECT getSetting('custom_a');
```

## See Also

- [Server Configuration Settings](#)

# Permissions for Queries

Queries in ClickHouse can be divided into several types:

1. Read data queries: `SELECT`, `SHOW`, `DESCRIBE`, `EXISTS`.
2. Write data queries: `INSERT`, `OPTIMIZE`.
3. Change settings query: `SET`, `USE`.
4. `DDL` queries: `CREATE`, `ALTER`, `RENAME`, `ATTACH`, `DETACH`, `DROP`, `TRUNCATE`.
5. `KILL QUERY`.

The following settings regulate user permissions by the type of query:

- `readonly` — Restricts permissions for all types of queries except DDL queries.
- `allow_ddl` — Restricts permissions for DDL queries.

`KILL QUERY` can be performed with any settings.

## `readonly`

Restricts permissions for reading data, write data and change settings queries.

See how the queries are divided into types [above](#).

Possible values:

- 0 — All queries are allowed.
- 1 — Only read data queries are allowed.
- 2 — Read data and change settings queries are allowed.

After setting `readonly = 1`, the user can't change `readonly` and `allow_ddl` settings in the current session.

When using the `GET` method in the [HTTP interface](#), `readonly = 1` is set automatically. To modify data, use the `POST` method.

Setting `readonly = 1` prohibit the user from changing all the settings. There is a way to prohibit the user from changing only specific settings, for details see [constraints on settings](#).

Default value: 0

## `allow_ddl`

Allows or denies `DDL` queries.

See how the queries are divided into types [above](#).

Possible values:

- 0 — DDL queries are not allowed.
- 1 — DDL queries are allowed.

You can't execute `SET allow_ddl = 1` if `allow_ddl = 0` for the current session.

Default value: 1

---

# Restrictions on Query Complexity

Restrictions on query complexity are part of the settings.

They are used to provide safer execution from the user interface.

Almost all the restrictions only apply to `SELECT`. For distributed query processing, restrictions are applied on each server separately.

ClickHouse checks the restrictions for data parts, not for each row. It means that you can exceed the value of restriction with the size of the data part.

Restrictions on the “maximum amount of something” can take the value 0, which means “unrestricted”. Most restrictions also have an ‘`overflow_mode`’ setting, meaning what to do when the limit is exceeded. It can take one of two values: `throw` or `break`. Restrictions on aggregation (`group_by_overflow_mode`) also have the value `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 do not add new keys to the set.

## max\_memory\_usage

The maximum amount of RAM to use for running a query on a single server.

In the default configuration file, the maximum is 10 GB.

The setting does not consider the volume of available memory or the total volume of memory on the machine.

The restriction applies to a single query within a single server.

You can use `SHOW PROCESSLIST` to see the current memory consumption for each query.

Besides, the peak memory consumption is tracked for each query and written to the log.

Memory usage is not monitored for the states of certain aggregate functions.

Memory usage is not fully tracked for states of the aggregate functions `min`, `max`, `any`, `anyLast`, `argMin`, `argMax` from `String` and `Array` arguments.

Memory consumption is also restricted by the parameters `max_memory_usage_for_user` and `max_server_memory_usage`.

## max\_memory\_usage\_for\_user

The maximum amount of RAM to use for running a user’s queries on a single server.

Default values are defined in [Settings.h](#). By default, the amount is not restricted (`max_memory_usage_for_user = 0`).

See also the description of [max\\_memory\\_usage](#).

## max\_rows\_to\_read

The following restrictions can be checked on each block (instead of on each row). That is, the restrictions can be broken a little.

A maximum number of rows that can be read from a table when running a query.

## max\_bytes\_to\_read

A maximum number of bytes (uncompressed data) that can be read from a table when running a query.

## `read_overflow_mode`

What to do when the volume of data read exceeds one of the limits: ‘throw’ or ‘break’. By default, throw.

## `max_rows_to_read_leaf`

The following restrictions can be checked on each block (instead of on each row). That is, the restrictions can be broken a little.

A maximum number of rows that can be read from a local table on a leaf node when running a distributed query. While distributed queries can issue a multiple sub-queries to each shard (leaf) - this limit will be checked only on the read stage on the leaf nodes and ignored on results merging stage on the root node. For example, cluster consists of 2 shards and each shard contains a table with 100 rows. Then distributed query which suppose to read all the data from both tables with setting `max_rows_to_read=150` will fail as in total it will be 200 rows. While query with `max_rows_to_read_leaf=150` will succeed since leaf nodes will read 100 rows at max.

## `max_bytes_to_read_leaf`

A maximum number of bytes (uncompressed data) that can be read from a local table on a leaf node when running a distributed query. While distributed queries can issue a multiple sub-queries to each shard (leaf) - this limit will be checked only on the read stage on the leaf nodes and ignored on results merging stage on the root node. For example, cluster consists of 2 shards and each shard contains a table with 100 bytes of data. Then distributed query which suppose to read all the data from both tables with setting `max_bytes_to_read=150` will fail as in total it will be 200 bytes. While query with `max_bytes_to_read_leaf=150` will succeed since leaf nodes will read 100 bytes at max.

## `read_overflow_mode_leaf`

What to do when the volume of data read exceeds one of the leaf limits: ‘throw’ or ‘break’. By default, throw.

## `max_rows_to_group_by`

A maximum number of unique keys received from aggregation. This setting lets you limit memory consumption when aggregating.

## `group_by_overflow_mode`

What to do when the number of unique keys for aggregation exceeds the limit: ‘throw’, ‘break’, or ‘any’. By default, throw.

Using the ‘any’ value lets you run an approximation of GROUP BY. The quality of this approximation depends on the statistical nature of the data.

## `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.

## max\_rows\_to\_sort

A maximum number of rows before sorting. This allows you to limit memory consumption when sorting.

## max\_bytes\_to\_sort

A maximum number of bytes before sorting.

## sort\_overflow\_mode

What to do if the number of rows received before sorting exceeds one of the limits: ‘throw’ or ‘break’. By default, throw.

## max\_result\_rows

Limit on the number of rows in the result. Also checked for subqueries, and on remote servers when running parts of a distributed query.

## max\_result\_bytes

Limit on the number of bytes in the result. The same as the previous setting.

## result\_overflow\_mode

What to do if the volume of the result exceeds one of the limits: ‘throw’ or ‘break’. By default, throw.

Using ‘break’ is similar to using LIMIT. Break interrupts execution only at the block level. This means that amount of returned rows is greater than max\_result\_rows, multiple of max\_block\_size and depends on max\_threads.

Example:

```
SET max_threads = 3, max_block_size = 3333;
SET max_result_rows = 3334, result_overflow_mode = 'break';

SELECT *
FROM numbers_mt(100000)
FORMAT Null;
```

Result:

```
6666 rows in set. ...
```

## max\_execution\_time

Maximum query execution time in seconds.

At this time, it is not checked for one of the sorting stages, or when merging and finalizing aggregate functions.

## timeout\_overflow\_mode

What to do if the query is run longer than ‘max\_execution\_time’: ‘throw’ or ‘break’. By default, throw.

## **min\_execution\_speed**

Minimal execution speed in rows per second. Checked on every data block when ‘timeout\_before\_checking\_execution\_speed’ expires. If the execution speed is lower, an exception is thrown.

## **min\_execution\_speed\_bytes**

A minimum number of execution bytes per second. Checked on every data block when ‘timeout\_before\_checking\_execution\_speed’ expires. If the execution speed is lower, an exception is thrown.

## **max\_execution\_speed**

A maximum number of execution rows per second. Checked on every data block when ‘timeout\_before\_checking\_execution\_speed’ expires. If the execution speed is high, the execution speed will be reduced.

## **max\_execution\_speed\_bytes**

A maximum number of execution bytes per second. Checked on every data block when ‘timeout\_before\_checking\_execution\_speed’ expires. If the execution speed is high, the execution speed will be reduced.

## **timeout\_before\_checking\_execution\_speed**

Checks that execution speed is not too slow (no less than ‘min\_execution\_speed’), after the specified time in seconds has expired.

## **max\_columns\_to\_read**

A maximum number of columns that can be read from a table in a single query. If a query requires reading a greater number of columns, it throws an exception.

## **max\_temporary\_columns**

A maximum number of temporary columns that must be kept in RAM at the same time when running a query, including constant columns. If there are more temporary columns than this, it throws an exception.

## **max\_temporary\_non\_const\_columns**

The same thing as ‘max\_temporary\_columns’, but without counting constant columns. Note that constant columns are formed fairly often when running a query, but they require approximately zero computing resources.

## **max\_subquery\_depth**

Maximum nesting depth of subqueries. If subqueries are deeper, an exception is thrown. By default, 100.

## **max\_pipeline\_depth**

Maximum pipeline depth. Corresponds to the number of transformations that each data block goes through during query processing. Counted within the limits of a single server. If the pipeline depth is greater, an exception is thrown. By default, 1000.

## **max\_ast\_depth**

Maximum nesting depth of a query syntactic tree. If exceeded, an exception is thrown. At this time, it isn't checked during parsing, but only after parsing the query. That is, a syntactic tree that is too deep can be created during parsing, but the query will fail. By default, 1000.

## max\_ast\_elements

A maximum number of elements in a query syntactic tree. If exceeded, an exception is thrown. In the same way as the previous setting, it is checked only after parsing the query. By default, 50,000.

## max\_rows\_in\_set

A maximum number of rows for a data set in the IN clause created from a subquery.

## max\_bytes\_in\_set

A maximum number of bytes (uncompressed data) used by a set in the IN clause created from a subquery.

## set\_overflow\_mode

What to do when the amount of data exceeds one of the limits: 'throw' or 'break'. By default, throw.

## max\_rows\_in\_distinct

A maximum number of different rows when using DISTINCT.

## max\_bytes\_in\_distinct

A maximum number of bytes used by a hash table when using DISTINCT.

## distinct\_overflow\_mode

What to do when the amount of data exceeds one of the limits: 'throw' or 'break'. By default, throw.

## max\_rows\_to\_transfer

A maximum number of rows that can be passed to a remote server or saved in a temporary table when using GLOBAL IN.

## max\_bytes\_to\_transfer

A maximum number of bytes (uncompressed data) that can be passed to a remote server or saved in a temporary table when using GLOBAL IN.

## transfer\_overflow\_mode

What to do when the amount of data exceeds one of the limits: 'throw' or 'break'. By default, throw.

## 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 does not throw an exception.

Default value: `THROW`.

## See Also

- [JOIN clause](#)
- [Join table engine](#)

## max\_partitions\_per\_insert\_block

Limits the maximum number of partitions in a single inserted block.

- Positive integer.
- 0 — Unlimited number of partitions.

Default value: 100.

## Details

When inserting data, ClickHouse calculates the number of partitions in the inserted block. If the number of partitions is more than `max_partitions_per_insert_block`, ClickHouse throws an exception with the following text:

"Too many partitions for single INSERT block (more than" + `toString(max_parts)` + "). The limit is controlled by 'max\_partitions\_per\_insert\_block' setting. A large number of partitions is a common misconception. It will lead to severe negative performance impact, including slow server startup, slow INSERT queries and slow SELECT queries. Recommended total number of partitions for a table is under 1000..10000. Please note, that partitioning is not intended to speed up SELECT queries (ORDER BY key is sufficient to make range queries fast). Partitions are intended for data manipulation (DROP PARTITION, etc)."'

## Settings Profiles

A settings profile is a collection of settings grouped under the same name.

### Information

ClickHouse also supports **SQL-driven workflow** for managing settings profiles. We recommend using it.

The profile can have any name. You can specify the same profile for different users. The most important thing you can write in the settings profile is `readonly=1`, which ensures read-only access.

Settings profiles can inherit from each other. To use inheritance, indicate one or multiple `profile` settings before the other settings that are listed in the profile. In case when one setting is defined in different profiles, the latest defined is used.

To apply all the settings in a profile, set the `profile` setting.

Example:

Install the `web` profile.

```
SET profile = 'web'
```

Settings profiles are declared in the user config file. This is usually `users.xml`.

Example:

```

<!-- 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>10000000000</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>

```

The example specifies two profiles: `default` and `web`.

The `default` profile has a special purpose: it must always be present and is applied when starting the server. In other words, the `default` profile contains default settings.

The `web` profile is a regular profile that can be set using the `SET` query or using a URL parameter in an HTTP query.

## Constraints on Settings

The constraints on settings can be defined in the `profiles` section of the `user.xml` configuration file and prohibit users from changing some of the settings with the `SET` query.

The constraints are defined as the following:

```

<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>

```

If the user tries to violate the constraints an exception is thrown and the setting isn't changed. There are supported three types of constraints: `min`, `max`, `readonly`. The `min` and `max` constraints specify upper and lower boundaries for a numeric setting and can be used in combination. The `readonly` constraint specifies that the user cannot change the corresponding setting at all.

**Example:** Let `users.xml` includes lines:

```

<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>

```

The following queries all throw exceptions:

```

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 5000000000.
Code: 452, e.displayText() = DB::Exception: Setting force_index_by_date should not be changed.

```

**Note:** the default profile has special handling: all the constraints defined for the default profile become the default constraints, so they restrict all the users until they're overridden explicitly for these users.

## User Settings

The `users` section of the `user.xml` configuration file contains user settings.

# Information

ClickHouse also supports **SQL-driven workflow** for managing users. We recommend using it.

Structure of the users section:

```
<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>
    <default_database>default</default_database>
    <databases>
      <database_name>
        <table_name>
          <filter>expression</filter>
        <table_name>
      </database_name>
    </databases>
  </user_name>
  <!-- Other users settings -->
</users>
```

## user\_name/password

Password can be specified in plaintext or in SHA256 (hex format).

- To assign a password in plaintext (**not recommended**), place it in a `password` element.

For example, `<password>qwerty</password>`. The password can be left blank.

- To assign a password using its SHA256 hash, place it in a `password_sha256_hex` element.

For example,

```
<password_sha256_hex>65e84be33532fb784c48129675f9eff3a682b27168c0ea744b2cf58ee02337c5</password_sha256_hex>
```

Example of how to generate a password from shell:

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

The first line of the result is the password. The second line is the corresponding SHA256 hash.

- For compatibility with MySQL clients, password can be specified in double SHA1 hash. Place it in `password_double_sha1_hex` element.

For example,

```
<password_double_sha1_hex>08b4a0f1de6ad37da17359e592c8d74788a83eb0</password_double_sha1_hex>.
```

Example of how to generate a password from shell:

```
PASSWORD=$(base64 < /dev/urandom | head -c8); echo "$PASSWORD"; echo -n "$PASSWORD" | sha1sum | tr -d '-' | xxd -r -p | sha1sum | tr -d '-'
```

The first line of the result is the password. The second line is the corresponding double SHA1 hash.

## access\_management

This setting enables or disables using of SQL-driven **access control and account management** for the user.

Possible values:

- 0 — Disabled.
- 1 — Enabled.

Default value: 0.

## user\_name/networks

List of networks from which the user can connect to the ClickHouse server.

Each element of the list can have one of the following forms:

- `<ip>` — IP address or network mask.

Examples: `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.

Example: `example01.host.ru`.

To check access, a DNS query is performed, and all returned IP addresses are compared to the peer address.

- `<host_regex>` — Regular expression for hostnames.

Example, `^example\d\d-\d\d\.\host\.ru$`

To check access, a **DNS PTR query** is performed for the peer address and then the specified regexp is applied. Then, another DNS query is performed for the results of the PTR query and all the received addresses are compared to the peer address. We strongly recommend that regexp ends with \$.

All results of DNS requests are cached until the server restarts.

## Examples

To open access for user from any network, specify:

```
<ip>::/0</ip>
```

## Warning

It's insecure to open access from any network unless you have a firewall properly configured or the server is not directly connected to Internet.

To open access only from localhost, specify:

```
<ip>::1</ip>
<ip>127.0.0.1</ip>
```

## user\_name/profile

You can assign a settings profile for the user. Settings profiles are configured in a separate section of the `users.xml` file. For more information, see [Profiles of Settings](#).

## user\_name/quota

Quotas allow you to track or limit resource usage over a period of time. Quotas are configured in the `quotas` section of the `users.xml` configuration file.

You can assign a quotas set for the user. For a detailed description of quotas configuration, see [Quotas](#).

## user\_name/databases

In this section, you can limit rows that are returned by ClickHouse for `SELECT` queries made by the current user, thus implementing basic row-level security.

### Example

The following configuration forces that user `user1` can only see the rows of `table1` as the result of `SELECT` queries, where the value of the `id` field is 1000.

```
<user1>
  <databases>
    <database_name>
      <table1>
        <filter>id = 1000</filter>
      </table1>
    </database_name>
  </databases>
</user1>
```

The `filter` can be any expression resulting in a `UInt8`-type value. It usually contains comparisons and logical operators. Rows from `database_name.table1` where filter results to 0 are not returned for this user. The filtering is incompatible with `PREWHERE` operations and disables `WHERE→PREWHERE` optimization.

## MergeTree tables settings

The values of `merge_tree` settings (for all MergeTree tables) can be viewed in the table `system.merge_tree_settings`, they can be overridden in `config.xml` in the `merge_tree` section, or set in the `SETTINGS` section of each table.

Override example in `config.xml`:

```
<merge_tree>
  <max_suspicious_broken_parts>5</max_suspicious_broken_parts>
</merge_tree>
```

An example to set in `SETTINGS` for a particular table:

```
CREATE TABLE foo
(
    `A` Int64
)
ENGINE = MergeTree
ORDER BY tuple()
SETTINGS max_suspicious_broken_parts = 500;
```

An example of changing the settings for a specific table with the `ALTER TABLE ... MODIFY SETTING` command:

```
ALTER TABLE foo
MODIFY SETTING max_suspicious_broken_parts = 100;
```

## parts\_to\_throw\_insert

If the number of active parts in a single partition exceeds the `parts_to_throw_insert` value, `INSERT` is interrupted with the `Too many parts (N)`. Merges are processing significantly slower than `inserts` exception.

Possible values:

- Any positive integer.

Default value: 300.

To achieve maximum performance of `SELECT` queries, it is necessary to minimize the number of parts processed, see [Merge Tree](#).

You can set a larger value to 600 (1200), this will reduce the probability of the `Too many parts` error, but at the same time `SELECT` performance might degrade. Also in case of a merge issue (for example, due to insufficient disk space) you will notice it later than it could be with the original 300.

## parts\_to\_delay\_insert

If the number of active parts in a single partition exceeds the `parts_to_delay_insert` value, an `INSERT` artificially slows down.

Possible values:

- Any positive integer.

Default value: 150.

ClickHouse artificially executes `INSERT` longer (adds 'sleep') so that the background merge process can merge parts faster than they are added.

## inactive\_parts\_to\_throw\_insert

If the number of inactive parts in a single partition more than the `inactive_parts_to_throw_insert` value, `INSERT` is interrupted with the "Too many inactive parts (N)". Parts cleaning are processing significantly slower than `inserts`" exception.

Possible values:

- Any positive integer.

Default value: 0 (unlimited).

## inactive\_parts\_to\_delay\_insert

If the number of inactive parts in a single partition in the table at least that many the `inactive_parts_to_delay_insert` value, an `INSERT` artificially slows down. It is useful when a server fails to clean up parts quickly enough.

Possible values:

- Any positive integer.

Default value: 0 (unlimited).

## max\_delay\_to\_insert

The value in seconds, which is used to calculate the `INSERT` delay, if the number of active parts in a single partition exceeds the `parts_to_delay_insert` value.

Possible values:

- Any positive integer.

Default value: 1.

The delay (in milliseconds) for `INSERT` is calculated by the formula:

```
max_k = parts_to_throw_insert - parts_to_delay_insert
k = 1 + parts_count_in_partition - parts_to_delay_insert
delay_milliseconds = pow(max_delay_to_insert * 1000, k / max_k)
```

For example if a partition has 299 active parts and `parts_to_throw_insert` = 300, `parts_to_delay_insert` = 150, `max_delay_to_insert` = 1, `INSERT` is delayed for `pow( 1 * 1000, (1 + 299 - 150) / (300 - 150) ) = 1000` milliseconds.

## max\_parts\_in\_total

If the total number of active parts in all partitions of a table exceeds the `max_parts_in_total` value `INSERT` is interrupted with the `Too many parts (N)` exception.

Possible values:

- Any positive integer.

Default value: 100000.

A large number of parts in a table reduces performance of ClickHouse queries and increases ClickHouse boot time. Most often this is a consequence of an incorrect design (mistakes when choosing a partitioning strategy - too small partitions).

## replicated\_deduplication\_window

The number of most recently inserted blocks for which Zookeeper stores hash sums to check for duplicates.

Possible values:

- Any positive integer.
- 0 (disable deduplication)

Default value: 100.

The `Insert` command creates one or more blocks (parts). When inserting into Replicated tables, ClickHouse for `insert deduplication` writes the hash sums of the created parts into Zookeeper. Hash sums are stored only for the most recent `replicated_deduplication_window` blocks. The oldest hash sums are removed from Zookeeper.

A large number of `replicated_deduplication_window` slows down `Inserts` because it needs to compare more entries.

The hash sum is calculated from the composition of the field names and types and the data of the inserted part (stream of bytes).

## `non_replicated_deduplication_window`

The number of the most recently inserted blocks in the non-replicated `MergeTree` table for which hash sums are stored to check for duplicates.

Possible values:

- Any positive integer.
- 0 (disable deduplication).

Default value: 0.

A deduplication mechanism is used, similar to replicated tables (see `replicated_deduplication_window` setting). The hash sums of the created parts are written to a local file on a disk.

## `replicated_deduplication_window_seconds`

The number of seconds after which the hash sums of the inserted blocks are removed from Zookeeper.

Possible values:

- Any positive integer.

Default value: 604800 (1 week).

Similar to `replicated_deduplication_window`, `replicated_deduplication_window_seconds` specifies how long to store hash sums of blocks for insert deduplication. Hash sums older than `replicated_deduplication_window_seconds` are removed from Zookeeper, even if they are less than `replicated_deduplication_window`.

## `replicated_fetches_http_connection_timeout`

HTTP connection timeout (in seconds) for part fetch requests. Inherited from default profile `http_connection_timeout` if not set explicitly.

Possible values:

- Any positive integer.
- 0 - Use value of `http_connection_timeout`.

Default value: 0.

## `replicated_fetches_http_send_timeout`

HTTP send timeout (in seconds) for part fetch requests. Inherited from default profile `http_send_timeout` if not set explicitly.

Possible values:

- Any positive integer.

- 0 - Use value of `http_send_timeout`.

Default value: 0.

## `replicated_fetches_http_receive_timeout`

HTTP receive timeout (in seconds) for fetch part requests. Inherited from default profile `http_receive_timeout` if not set explicitly.

Possible values:

- Any positive integer.
- 0 - Use value of `http_receive_timeout`.

Default value: 0.

## `max_replicated_fetches_network_bandwidth`

Limits the maximum speed of data exchange over the network in bytes per second for `replicated` fetches.

This setting is applied to a particular table, unlike the

`max_replicated_fetches_network_bandwidth_for_server` setting, which is applied to the server.

You can limit both server network and network for a particular table, but for this the value of the table-level setting should be less than server-level one. Otherwise the server considers only the `max_replicated_fetches_network_bandwidth_for_server` setting.

The setting isn't followed perfectly accurately.

Possible values:

- Positive integer.
- 0 — Unlimited.

Default value: 0.

### **Usage**

Could be used for throttling speed when replicating data to add or replace new nodes.

## `max_replicated_sends_network_bandwidth`

Limits the maximum speed of data exchange over the network in bytes per second for `replicated` sends.

This setting is applied to a particular table, unlike the

`max_replicated_sends_network_bandwidth_for_server` setting, which is applied to the server.

You can limit both server network and network for a particular table, but for this the value of the table-level setting should be less than server-level one. Otherwise the server considers only the `max_replicated_sends_network_bandwidth_for_server` setting.

The setting isn't followed perfectly accurately.

Possible values:

- Positive integer.
- 0 — Unlimited.

Default value: 0.

### **Usage**

Could be used for throttling speed when replicating data to add or replace new nodes.

## old\_parts\_lifetime

The time (in seconds) of storing inactive parts to protect against data loss during spontaneous server reboots.

Possible values:

- Any positive integer.

Default value: 480.

After merging several parts into a new part, ClickHouse marks the original parts as inactive and deletes them only after `old_parts_lifetime` seconds.

Inactive parts are removed if they are not used by current queries, i.e. if the `refcount` of the part is zero.

`fsync` is not called for new parts, so for some time new parts exist only in the server's RAM (OS cache). If the server is rebooted spontaneously, new parts can be lost or damaged.

To protect data inactive parts are not deleted immediately.

During startup ClickHouse checks the integrity of the parts.

If the merged part is damaged ClickHouse returns the inactive parts to the active list, and later merges them again. Then the damaged part is renamed (the `broken_` prefix is added) and moved to the `detached` folder.

If the merged part is not damaged, then the original inactive parts are renamed (the `ignored_` prefix is added) and moved to the `detached` folder.

The default `dirty_expire_centisecs` value (a Linux kernel setting) is 30 seconds (the maximum time that written data is stored only in RAM), but under heavy loads on the disk system data can be written much later. Experimentally, a value of 480 seconds was chosen for `old_parts_lifetime`, during which a new part is guaranteed to be written to disk.

## max\_bytes\_to\_merge\_at\_max\_space\_in\_pool

The maximum total parts size (in bytes) to be merged into one part, if there are enough resources available.

`max_bytes_to_merge_at_max_space_in_pool` -- roughly corresponds to the maximum possible part size created by an automatic background merge.

Possible values:

- Any positive integer.

Default value: 161061273600 (150 GB).

The merge scheduler periodically analyzes the sizes and number of parts in partitions, and if there is enough free resources in the pool, it starts background merges. Merges occur until the total size of the source parts is less than `max_bytes_to_merge_at_max_space_in_pool`.

Merges initiated by `OPTIMIZE FINAL` ignore `max_bytes_to_merge_at_max_space_in_pool` and merge parts only taking into account available resources (free disk's space) until one part remains in the partition.

## max\_bytes\_to\_merge\_at\_min\_space\_in\_pool

The maximum total part size (in bytes) to be merged into one part, with the minimum available resources in the background pool.

Possible values:

- Any positive integer.

Default value: 1048576 (1 MB)

`max_bytes_to_merge_at_min_space_in_pool` defines the maximum total size of parts which can be merged despite the lack of available disk space (in pool). This is necessary to reduce the number of small parts and the chance of `Too many parts` errors.

Merges book disk space by doubling the total merged parts sizes. Thus, with a small amount of free disk space, a situation may happen that there is free space, but this space is already booked by ongoing large merges, so other merges unable to start, and the number of small parts grows with every insert.

## merge\_max\_block\_size

The number of rows that are read from the merged parts into memory.

Possible values:

- Any positive integer.

Default value: 8192

Merge reads rows from parts in blocks of `merge_max_block_size` rows, then merges and writes the result into a new part. The read block is placed in RAM, so `merge_max_block_size` affects the size of the RAM required for the merge. Thus, merges can consume a large amount of RAM for tables with very wide rows (if the average row size is 100kb, then when merging 10 parts,  $(100\text{kb} * 10 * 8192) = \sim 8\text{GB}$  of RAM). By decreasing `merge_max_block_size`, you can reduce the amount of RAM required for a merge but slow down a merge.

## max\_part\_loading\_threads

The maximum number of threads that read parts when ClickHouse starts.

Possible values:

- Any positive integer.

Default value: auto (number of CPU cores).

During startup ClickHouse reads all parts of all tables (reads files with metadata of parts) to build a list of all parts in memory. In some systems with a large number of parts this process can take a long time, and this time might be shortened by increasing `max_part_loading_threads` (if this process is not CPU and disk I/O bound).

## max\_partitions\_to\_read

Limits the maximum number of partitions that can be accessed in one query.

The setting value specified when the table is created can be overridden via query-level setting.

Possible values:

- Any positive integer.

Default value: -1 (unlimited).

## allow\_floating\_point\_partition\_key

Enables to allow floating-point number as a partition key.

Possible values:

- 0 — Floating-point partition key not allowed.
- 1 — Floating-point partition key allowed.

Default value: 0.

## check\_sample\_column\_is\_correct

Enables the check at table creation, that the data type of a column for sampling or sampling expression is correct. The data type must be one of unsigned [integer types](#): UInt8, UInt16, UInt32, UInt64.

Possible values:

- true — The check is enabled.
- false — The check is disabled at table creation.

Default value: true.

By default, the ClickHouse server checks at table creation the data type of a column for sampling or sampling expression. If you already have tables with incorrect sampling expression and do not want the server to raise an exception during startup, set `check_sample_column_is_correct` to false.

---

## Settings

### distributed\_product\_mode

Changes the behaviour of [distributed subqueries](#).

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.

Restrictions:

- Only applied for IN and JOIN subqueries.
- Only if the FROM section uses a distributed table containing more than one shard.
- If the subquery concerns a distributed table containing more than one shard.
- Not used for a table-valued [remote](#) function.

Possible values:

- `deny` — Default value. Prohibits using these types of subqueries (returns the “Double-distributed in/JION subqueries is denied” exception).
- `local` — Replaces the database and table in the subquery with local ones for the destination server (shard), leaving the normal IN/JION.
- `global` — Replaces the IN/JION query with GLOBAL IN/GLOBAL JOIN.
- `allow` — Allows the use of these types of subqueries.

### prefer\_global\_in\_and\_join

Enables the replacement of IN/JION operators with GLOBAL IN/GLOBAL JOIN.

Possible values:

- 0 — Disabled. IN/JION operators are not replaced with GLOBAL IN/GLOBAL JOIN.

- 1 — Enabled. `IN/JOIN` operators are replaced with `GLOBAL IN/GLOBAL JOIN`.

Default value: 0.

## Usage

Although `SET distributed_product_mode=global` can change the queries behavior for the distributed tables, it's not suitable for local tables or tables from external resources. Here is when the `prefer_global_in_and_join` setting comes into play.

For example, we have query serving nodes that contain local tables, which are not suitable for distribution. We need to scatter their data on the fly during distributed processing with the `GLOBAL` keyword — `GLOBAL IN/GLOBAL JOIN`.

Another use case of `prefer_global_in_and_join` is accessing tables created by external engines. This setting helps to reduce the number of calls to external sources while joining such tables: only one call per query.

## See also:

- [Distributed subqueries](#) for more information on how to use `GLOBAL IN/GLOBAL JOIN`

## enable\_optimize\_predicate\_expression

Turns on predicate pushdown in `SELECT` queries.

Predicate pushdown may significantly reduce network traffic for distributed queries.

Possible values:

- 0 — Disabled.
- 1 — Enabled.

Default value: 1.

## Usage

Consider the following queries:

1. `SELECT count() FROM test_table WHERE date = '2018-10-10'`
2. `SELECT count() FROM (SELECT * FROM test_table) WHERE date = '2018-10-10'`

If `enable_optimize_predicate_expression = 1`, then the execution time of these queries is equal because ClickHouse applies `WHERE` to the subquery when processing it.

If `enable_optimize_predicate_expression = 0`, then the execution time of the second query is much longer because the `WHERE` clause applies to all the data after the subquery finishes.

## fallback\_to\_stale\_replicas\_for\_distributed\_queries

Forces a query to an out-of-date replica if updated data is not available. See [Replication](#).

ClickHouse selects the most relevant from the outdated replicas of the table.

Used when performing `SELECT` from a distributed table that points to replicated tables.

By default, 1 (enabled).

## force\_index\_by\_date

Disables query execution if the index can't be used by date.

Works with tables in the MergeTree family.

If `force_index_by_date=1`, ClickHouse checks whether the query has a date key condition that can be used for restricting data ranges. If there is no suitable condition, it throws an exception. However, it does not check whether the condition reduces the amount of data to read. For example, the condition `Date != '2000-01-01'` is acceptable even when it matches all the data in the table (i.e., running the query requires a full scan). For more information about ranges of data in MergeTree tables, see [MergeTree](#).

## force\_primary\_key

Disables query execution if indexing by the primary key is not possible.

Works with tables in the MergeTree family.

If `force_primary_key=1`, ClickHouse checks to see if the query has a primary key condition that can be used for restricting data ranges. If there is no suitable condition, it throws an exception. However, it does not check whether the condition reduces the amount of data to read. For more information about data ranges in MergeTree tables, see [MergeTree](#).

## use\_skip\_indexes

Use data skipping indexes during query execution.

Possible values:

- 0 — Disabled.
- 1 — Enabled.

Default value: 1.

## force\_data\_skipping\_indices

Disables query execution if passed data skipping indices wasn't used.

Consider the following example:

```
CREATE TABLE data
(
    key Int,
    d1 Int,
    d1_null Nullable(Int),
    INDEX d1_idx d1 TYPE minmax GRANULARITY 1,
    INDEX d1_null_idx assumeNotNull(d1_null) TYPE minmax GRANULARITY 1
)
Engine=MergeTree()
ORDER BY key;

SELECT * FROM data_01515;
SELECT * FROM data_01515 SETTINGS force_data_skipping_indices=""; -- query will produce CANNOT_PARSE_TEXT error.
SELECT * FROM data_01515 SETTINGS force_data_skipping_indices='d1_idx'; -- query will produce INDEX_NOT_USED error.
SELECT * FROM data_01515 WHERE d1 = 0 SETTINGS force_data_skipping_indices='d1_idx'; -- Ok.
SELECT * FROM data_01515 WHERE d1 = 0 SETTINGS force_data_skipping_indices='`d1_idx`'; -- Ok (example of full featured parser).
SELECT * FROM data_01515 WHERE d1 = 0 SETTINGS force_data_skipping_indices='`d1_idx` , d1_null_idx'; -- query will produce INDEX_NOT_USED error, since d1_null_idx is not used.
SELECT * FROM data_01515 WHERE d1 = 0 AND assumeNotNull(d1_null) = 0 SETTINGS
force_data_skipping_indices='`d1_idx` , d1_null_idx'; -- Ok.
```

Works with tables in the MergeTree family.

## format\_schema

This parameter is useful when you are using formats that require a schema definition, such as [Cap'n Proto](#) or [Protobuf](#). The value depends on the format.

## fsync\_metadata

Enables or disables [fsync](#) when writing `.sql` files. Enabled by default.

It makes sense to disable it if the server has millions of tiny tables that are constantly being created and destroyed.

## function\_range\_max\_elements\_in\_block

Sets the safety threshold for data volume generated by function [range](#). Defines the maximum number of values generated by function per block of data (sum of array sizes for every row in a block).

Possible values:

- Positive integer.

Default value: 500,000,000.

### See Also

- [max\\_block\\_size](#)
- [min\\_insert\\_block\\_size\\_rows](#)

## enable\_http\_compression

Enables or disables data compression in the response to an HTTP request.

For more information, read the [HTTP interface description](#).

Possible values:

- 0 — Disabled.
- 1 — Enabled.

Default value: 0.

## http\_zlib\_compression\_level

Sets the level of data compression in the response to an HTTP request if [enable\\_http\\_compression = 1](#).

Possible values: Numbers from 1 to 9.

Default value: 3.

## http\_native\_compression\_disable\_checksumming\_on\_decompression

Enables or disables checksum verification when decompressing the HTTP POST data from the client. Used only for ClickHouse native compression format (not used with `gzip` or `deflate`).

For more information, read the [HTTP interface description](#).

Possible values:

- 0 — Disabled.
- 1 — Enabled.

Default value: 0.

## http\_max\_uri\_size

Sets the maximum URI length of an HTTP request.

Possible values:

- Positive integer.

Default value: 1048576.

## table\_function\_remote\_max\_addresses

Sets the maximum number of addresses generated from patterns for the `remote` function.

Possible values:

- Positive integer.

Default value: 1000.

## glob\_expansion\_max\_elements

Sets the maximum number of addresses generated from patterns for external storages and table functions (like `url`) except the `remote` function.

Possible values:

- Positive integer.

Default value: 1000.

## send\_progress\_in\_http\_headers

Enables or disables `X-ClickHouse-Progress` HTTP response headers in `clickhouse-server` responses.

For more information, read the [HTTP interface description](#).

Possible values:

- 0 — Disabled.
- 1 — Enabled.

Default value: 0.

## max\_http\_get\_redirects

Limits the maximum number of HTTP GET redirect hops for `URL`-engine tables. The setting applies to both types of tables: those created by the `CREATE TABLE` query and by the `url` table function.

Possible values:

- Any positive integer number of hops.
- 0 — No hops allowed.

Default value: 0.

## input\_format\_allow\_errors\_num

Sets the maximum number of acceptable errors when reading from text formats (CSV, TSV, etc.).

The default value is 0.

Always pair it with `input_format_allow_errors_ratio`.

If an error occurred while reading rows but the error counter is still less than `input_format_allow_errors_num`, ClickHouse ignores the row and moves on to the next one.

If both `input_format_allow_errors_num` and `input_format_allow_errors_ratio` are exceeded, ClickHouse throws an exception.

## `input_format_allow_errors_ratio`

Sets the maximum percentage of errors allowed when reading from text formats (CSV, TSV, etc.).

The percentage of errors is set as a floating-point number between 0 and 1.

The default value is 0.

Always pair it with `input_format_allow_errors_num`.

If an error occurred while reading rows but the error counter is still less than `input_format_allow_errors_ratio`, ClickHouse ignores the row and moves on to the next one.

If both `input_format_allow_errors_num` and `input_format_allow_errors_ratio` are exceeded, ClickHouse throws an exception.

## `input_format_parquet_import_nested`

Enables or disables the ability to insert the data into **Nested** columns as an array of structs in **Parquet** input format.

Possible values:

- 0 — Data can not be inserted into **Nested** columns as an array of structs.
- 1 — Data can be inserted into **Nested** columns as an array of structs.

Default value: 0.

## `input_format_arrow_import_nested`

Enables or disables the ability to insert the data into **Nested** columns as an array of structs in **Arrow** input format.

Possible values:

- 0 — Data can not be inserted into **Nested** columns as an array of structs.
- 1 — Data can be inserted into **Nested** columns as an array of structs.

Default value: 0.

## `input_format_orc_import_nested`

Enables or disables the ability to insert the data into **Nested** columns as an array of structs in **ORC** input format.

Possible values:

- 0 — Data can not be inserted into **Nested** columns as an array of structs.

- 1 — Data can be inserted into `Nested` columns as an array of structs.

Default value: 0.

## input\_format\_values\_interpret\_expressions

Enables or disables the full SQL parser if the fast stream parser can't parse the data. This setting is used only for the `Values` format at the data insertion. For more information about syntax parsing, see the [Syntax](#) section.

Possible values:

- 0 — Disabled.

In this case, you must provide formatted data. See the [Formats](#) section.

- 1 — Enabled.

In this case, you can use an SQL expression as a value, but data insertion is much slower this way. If you insert only formatted data, then ClickHouse behaves as if the setting value is 0.

Default value: 1.

### Example of Use

Insert the `DateTime` type value with the different settings.

```
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.

The last query is equivalent to the following:

```
SET input_format_values_interpret_expressions = 0;
INSERT INTO datetime_t SELECT now()
```

Ok.

## input\_format\_values\_deduce\_templates\_of\_expressions

Enables or disables template deduction for SQL expressions in `Values` format. It allows parsing and interpreting expressions in `Values` much faster if expressions in consecutive rows have the same structure. ClickHouse tries to deduce the template of an expression, parse the following rows using this template and evaluate the expression on a batch of successfully parsed rows.

Possible values:

- 0 — Disabled.

- 1 — Enabled.

Default value: 1.

For the following query:

```
INSERT INTO test VALUES (lower('Hello')), (lower('world')), (lower('INSERT')), (upper('Values')), ...
```

- If `input_format_values_interpret_expressions=1` and `format_values_deduce_templates_of_expressions=0`, expressions are interpreted separately for each row (this is very slow for large number of rows).
- If `input_format_values_interpret_expressions=0` and `format_values_deduce_templates_of_expressions=1`, expressions in the first, second and third rows are parsed using template `lower(String)` and interpreted together, expression in the forth row is parsed with another template (`upper(String)`).
- If `input_format_values_interpret_expressions=1` and `format_values_deduce_templates_of_expressions=1`, the same as in previous case, but also allows fallback to interpreting expressions separately if it's not possible to deduce template.

## input\_format\_values\_accurate\_types\_of\_literals

This setting is used only when `input_format_values_deduce_templates_of_expressions = 1`. Expressions for some column may have the same structure, but contain numeric literals of different types, e.g.

```
(..., abs(0), ...),      -- UInt64 literal  
(..., abs(3.141592654), ...), -- Float64 literal  
(..., abs(-1), ...),     -- Int64 literal
```

Possible values:

- 0 — Disabled.

In this case, ClickHouse may use a more general type for some literals (e.g., `Float64` or `Int64` instead of `UInt64` for `42`), but it may cause overflow and precision issues.

- 1 — Enabled.

In this case, ClickHouse checks the actual type of literal and uses an expression template of the corresponding type. In some cases, it may significantly slow down expression evaluation in `Values`.

Default value: 1.

## input\_format\_defaults\_for\_omitted\_fields

When performing `INSERT` queries, replace omitted input column values with default values of the respective columns. This option only applies to `JSONEachRow`, `CSV` and `TabSeparated` formats.

### Note

When this option is enabled, extended table metadata are sent from server to client. It consumes additional computing resources on the server and can reduce performance.

Possible values:

- 0 — Disabled.
- 1 — Enabled.

Default value: 1.

## input\_format\_tsv\_empty\_as\_default

When enabled, replace empty input fields in TSV with default values. For complex default expressions `input_format_defaults_for_omitted_fields` must be enabled too.

Disabled by default.

## input\_format\_tsv\_enum\_as\_number

Enables or disables parsing enum values as enum ids for TSV input format.

Possible values:

- 0 — Enum values are parsed as values.
- 1 — Enum values are parsed as enum IDs.

Default value: 0.

### Example

Consider the table:

```
CREATE TABLE table_with_enum_column_for_tsv_insert (Id Int32,Value Enum('first' = 1,'second' = 2))
ENGINE=Memory();
```

When the `input_format_tsv_enum_as_number` setting is enabled:

```
SET input_format_tsv_enum_as_number = 1;
INSERT INTO table_with_enum_column_for_tsv_insert FORMAT TSV 102  2;
INSERT INTO table_with_enum_column_for_tsv_insert FORMAT TSV 103  1;
SELECT * FROM table_with_enum_column_for_tsv_insert;
```

Result:

Id	Value
102	second
103	first

When the `input_format_tsv_enum_as_number` setting is disabled, the `INSERT` query:

```
SET input_format_tsv_enum_as_number = 0;
INSERT INTO table_with_enum_column_for_tsv_insert FORMAT TSV 102  2;
```

throws an exception.

## input\_format\_null\_as\_default

Enables or disables the initialization of `NULL` fields with **default values**, if data type of these fields is not **nullable**.

If column type is not nullable and this setting is disabled, then inserting `NULL` causes an exception. If column type is nullable, then `NULL` values are inserted as is, regardless of this setting.

This setting is applicable to `INSERT ... VALUES` queries for text input formats.

Possible values:

- 0 — Inserting `NULL` into a not nullable column causes an exception.
- 1 — `NULL` fields are initialized with default column values.

Default value: 1.

## insert\_null\_as\_default

Enables or disables the insertion of **default values** instead of `NULL` into columns with not **nullable** data type. If column type is not nullable and this setting is disabled, then inserting `NULL` causes an exception. If column type is nullable, then `NULL` values are inserted as is, regardless of this setting.

This setting is applicable to `INSERT ... SELECT` queries. Note that `SELECT` subqueries may be concatenated with `UNION ALL` clause.

Possible values:

- 0 — Inserting `NULL` into a not nullable column causes an exception.
- 1 — Default column value is inserted instead of `NULL`.

Default value: 1.

## input\_format\_skip\_unknown\_fields

Enables or disables skipping insertion of extra data.

When writing data, ClickHouse throws an exception if input data contain columns that do not exist in the target table. If skipping is enabled, ClickHouse does not insert extra data and does not throw an exception.

Supported formats:

- [JSONEachRow](#)
- [CSVWithNames](#)
- [TabSeparatedWithNames](#)
- [TSKV](#)

Possible values:

- 0 — Disabled.
- 1 — Enabled.

Default value: 0.

## input\_format\_import\_nested\_json

Enables or disables the insertion of JSON data with nested objects.

Supported formats:

- [JSONEachRow](#)

Possible values:

- 0 — Disabled.
- 1 — Enabled.

Default value: 0.

See also:

- [Usage of Nested Structures](#) with the `JSONEachRow` format.

## input\_format\_with\_names\_use\_header

Enables or disables checking the column order when inserting data.

To improve insert performance, we recommend disabling this check if you are sure that the column order of the input data is the same as in the target table.

Supported formats:

- [CSVWithNames](#)
- [TabSeparatedWithNames](#)

Possible values:

- 0 — Disabled.
- 1 — Enabled.

Default value: 1.

## date\_time\_input\_format

Allows choosing a parser of the text representation of date and time.

The setting does not apply to [date and time functions](#).

Possible values:

- `'best_effort'` — Enables extended parsing.

ClickHouse can parse the basic `YYYY-MM-DD HH:MM:SS` format and all [ISO 8601](#) date and time formats. For example, `'2018-06-08T01:02:03.000Z'`.

- `'basic'` — Use basic parser.

ClickHouse can parse only the basic `YYYY-MM-DD HH:MM:SS` or `YYYY-MM-DD` format. For example, `2019-08-20 10:18:56` or `2019-08-20`.

Default value: `'basic'`.

See also:

- [DateTime data type](#).
- [Functions for working with dates and times](#).

## date\_time\_output\_format

Allows choosing different output formats of the text representation of date and time.

Possible values:

- `simple` - Simple output format.

ClickHouse output date and time `YYYY-MM-DD hh:mm:ss` format. For example, `2019-08-20 10:18:56`. The calculation is performed according to the data type's time zone (if present) or server time zone.

- `iso` - ISO output format.

ClickHouse output date and time in [ISO 8601](#) `YYYY-MM-DDThh:mm:ssZ` format. For example, `2019-08-20T10:18:56Z`. Note that output is in UTC (`Z` means UTC).

- `unix_timestamp` - Unix timestamp output format.

ClickHouse output date and time in [Unix timestamp](#) format. For example `1566285536`.

Default value: `simple`.

See also:

- [DateTime data type](#).
- [Functions for working with dates and times](#).

## join\_default\_strictness

Sets default strictness for [JOIN clauses](#).

Possible values:

- `ALL` — If the right table has several matching rows, ClickHouse creates a [Cartesian product](#) from matching rows. This is the normal `JOIN` behaviour from standard SQL.
- `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` and `ALL` are the same.
- `ASOF` — For joining sequences with an uncertain match.
- `Empty string` — If `ALL` or `ANY` is not specified in the query, ClickHouse throws an exception.

Default value: `ALL`.

## join\_algorithm

Specifies `JOIN` algorithm.

Possible values:

- `hash` — [Hash join algorithm](#) is used.
- `partial_merge` — [Sort-merge algorithm](#) is used.
- `prefer_partial_merge` — ClickHouse always tries to use `merge join` if possible.
- `auto` — ClickHouse tries to change `hash join` to `merge join` on the fly to avoid out of memory.

Default value: `hash`.

When using `hash` algorithm the right part of `JOIN` is uploaded into RAM.

When using `partial_merge` algorithm ClickHouse sorts the data and dumps it to the disk. The `merge` algorithm in ClickHouse differs a bit from the classic realization. First ClickHouse sorts the right table by [join key](#) in blocks and creates min-max index for sorted blocks. Then it sorts parts of left table by `join key` and joins them over right table. The min-max index is also used to skip unneeded right table blocks.

## join\_any\_take\_last\_row

Changes behaviour of join operations with `ANY` strictness.

## Attention

This setting applies only for `JOIN` operations with **Join** engine tables.

Possible values:

- 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.

Default value: 0.

See also:

- [JOIN clause](#)
- [Join table engine](#)
- [join\\_default\\_strictness](#)

## join\_use\_nulls

Sets the type of `JOIN` behaviour. When merging tables, empty cells may appear. ClickHouse fills them differently based on this setting.

Possible values:

- 0 — The empty cells are filled with the default value of the corresponding field type.
- 1 — `JOIN` behaves the same way as in standard SQL. The type of the corresponding field is converted to `Nullable`, and empty cells are filled with `NULL`.

Default value: 0.

## partial\_merge\_join\_optimizations

Disables optimizations in partial merge join algorithm for `JOIN` queries.

By default, this setting enables improvements that could lead to wrong results. If you see suspicious results in your queries, disable optimizations by this setting. Optimizations can be different in different versions of the ClickHouse server.

Possible values:

- 0 — Optimizations disabled.
- 1 — Optimizations enabled.

Default value: 1.

## partial\_merge\_join\_rows\_in\_right\_blocks

Limits sizes of right-hand join data blocks in partial merge join algorithm for `JOIN` queries.

ClickHouse server:

1. Splits right-hand join data into blocks with up to the specified number of rows.
2. Indexes each block with its minimum and maximum values.
3. Unloads prepared blocks to disk if it is possible.

Possible values:

- Any positive integer. Recommended range of values: [1000, 100000].

Default value: 65536.

## join\_on\_disk\_max\_files\_to\_merge

Limits the number of files allowed for parallel sorting in MergeJoin operations when they are executed on disk.

The bigger the value of the setting, the more RAM used and the less disk I/O needed.

Possible values:

- Any positive integer, starting from 2.

Default value: 64.

## any\_join\_distinct\_right\_table\_keys

Enables legacy ClickHouse server behaviour in ANY INNER|LEFT JOIN operations.

### Warning

Use this setting only for backward compatibility if your use cases depend on legacy JOIN behaviour.

When the legacy behaviour enabled:

- Results of t1 ANY LEFT JOIN t2 and t2 ANY RIGHT JOIN t1 operations are not equal because ClickHouse uses the logic with many-to-one left-to-right table keys mapping.
- Results of ANY INNER JOIN operations contain all rows from the left table like the SEMI LEFT JOIN operations do.

When the legacy behaviour disabled:

- Results of t1 ANY LEFT JOIN t2 and t2 ANY RIGHT JOIN t1 operations are equal because ClickHouse uses the logic which provides one-to-many keys mapping in ANY RIGHT JOIN operations.
- Results of ANY INNER JOIN operations contain one row per key from both the left and right tables.

Possible values:

- 0 — Legacy behaviour is disabled.
- 1 — Legacy behaviour is enabled.

Default value: 0.

See also:

- [JOIN strictness](#)

## temporary\_files\_codec

Sets compression codec for temporary files used in sorting and joining operations on disk.

Possible values:

- LZ4 — LZ4 compression is applied.
- NONE — No compression is applied.

Default value: LZ4.

## max\_block\_size

In ClickHouse, data is processed by blocks (sets of column parts). The internal processing cycles for a single block are efficient enough, but there are noticeable expenditures on each block. The `max_block_size` setting is a recommendation for what size of the block (in a count of rows) to load from tables. The block size shouldn't be too small, so that the expenditures on each block are still noticeable, but not too large so that the query with LIMIT that is completed after the first block is processed quickly. The goal is to avoid consuming too much memory when extracting a large number of columns in multiple threads and to preserve at least some cache locality.

Default value: 65,536.

Blocks the size of `max_block_size` are not always loaded from the table. If it is obvious that less data needs to be retrieved, a smaller block is processed.

## preferred\_block\_size\_bytes

Used for the same purpose as `max_block_size`, but it sets the recommended block size in bytes by adapting it to the number of rows in the block.

However, the block size cannot be more than `max_block_size` rows.

By default: 1,000,000. It only works when reading from MergeTree engines.

## merge\_tree\_min\_rows\_for\_concurrent\_read

If the number of rows to be read from a file of a [MergeTree](#) table exceeds `merge_tree_min_rows_for_concurrent_read` then ClickHouse tries to perform a concurrent reading from this file on several threads.

Possible values:

- Any positive integer.

Default value: 163840.

## merge\_tree\_min\_bytes\_for\_concurrent\_read

If the number of bytes to read from one file of a [MergeTree](#)-engine table exceeds `merge_tree_min_bytes_for_concurrent_read`, then ClickHouse tries to concurrently read from this file in several threads.

Possible value:

- Any positive integer.

Default value: 251658240.

## merge\_tree\_min\_rows\_for\_seek

If the distance between two data blocks to be read in one file is less than `merge_tree_min_rows_for_seek` rows, then ClickHouse does not seek through the file but reads the data sequentially.

Possible values:

- Any positive integer.

Default value: 0.

## merge\_tree\_min\_bytes\_for\_seek

If the distance between two data blocks to be read in one file is less than `merge_tree_min_bytes_for_seek` bytes, then ClickHouse sequentially reads a range of file that contains both blocks, thus avoiding extra seek.

Possible values:

- Any positive integer.

Default value: 0.

## merge\_tree\_coarse\_index\_granularity

When searching for data, ClickHouse checks the data marks in the index file. If ClickHouse finds that required keys are in some range, it divides this range into `merge_tree_coarse_index_granularity` subranges and searches the required keys there recursively.

Possible values:

- Any positive even integer.

Default value: 8.

## merge\_tree\_max\_rows\_to\_use\_cache

If ClickHouse should read more than `merge_tree_max_rows_to_use_cache` rows in one query, it does not use the cache of uncompressed blocks.

The cache of uncompressed blocks stores data extracted for queries. ClickHouse uses this cache to speed up responses to repeated small queries. This setting protects the cache from trashing by queries that read a large amount of data. The [uncompressed\\_cache\\_size](#) server setting defines the size of the cache of uncompressed blocks.

Possible values:

- Any positive integer.

Default value:  $128 \times 8192$ .

## merge\_tree\_max\_bytes\_to\_use\_cache

If ClickHouse should read more than `merge_tree_max_bytes_to_use_cache` bytes in one query, it does not use the cache of uncompressed blocks.

The cache of uncompressed blocks stores data extracted for queries. ClickHouse uses this cache to speed up responses to repeated small queries. This setting protects the cache from trashing by queries that read a large amount of data. The [uncompressed\\_cache\\_size](#) server setting defines the size of the cache of uncompressed blocks.

Possible values:

- Any positive integer.

Default value: 2013265920.

## merge\_tree\_clear\_old\_temporary\_directories\_interval\_seconds

Sets the interval in seconds for ClickHouse to execute the cleanup of old temporary directories.

Possible values:

- Any positive integer.

Default value: 60 seconds.

## merge\_tree\_clear\_old\_parts\_interval\_seconds

Sets the interval in seconds for ClickHouse to execute the cleanup of old parts, WALs, and mutations.

Possible values:

- Any positive integer.

Default value: 1 second.

## min\_bytes\_to\_use\_direct\_io

The minimum data volume required for using direct I/O access to the storage disk.

ClickHouse uses this setting when reading data from tables. If the total storage volume of all the data to be read exceeds `min_bytes_to_use_direct_io` bytes, then ClickHouse reads the data from the storage disk with the `O_DIRECT` option.

Possible values:

- 0 — Direct I/O is disabled.
- Positive integer.

Default value: 0.

## network\_compression\_method

Sets the method of data compression that is used for communication between servers and between server and [clickhouse-client](#).

Possible values:

- `LZ4` — sets LZ4 compression method.
- `ZSTD` — sets ZSTD compression method.

Default value: `LZ4`.

### See Also

- [network\\_zstd\\_compression\\_level](#)

## network\_zstd\_compression\_level

Adjusts the level of ZSTD compression. Used only when `network_compression_method` is set to `ZSTD`.

Possible values:

- Positive integer from 1 to 15.

Default value: 1.

## log\_queries

Setting up query logging.

Queries sent to ClickHouse with this setup are logged according to the rules in the `query_log` server configuration parameter.

Example:

```
log_queries=1
```

## log\_queries\_min\_query\_duration\_ms

If enabled (non-zero), queries faster than the value of this setting will not be logged (you can think about this as a `long_query_time` for [MySQL Slow Query Log](#)), and this basically means that you will not find them in the following tables:

- `system.query_log`
- `system.query_thread_log`

Only the queries with the following type will get to the log:

- `QUERY_FINISH`
- `EXCEPTION_WHILE_PROCESSING`
- Type: milliseconds
- Default value: 0 (any query)

## log\_queries\_min\_type

`query_log` minimal type to log.

Possible values:

- `QUERY_START` (=1)
- `QUERY_FINISH` (=2)
- `EXCEPTION_BEFORE_START` (=3)
- `EXCEPTION_WHILE_PROCESSING` (=4)

Default value: `QUERY_START`.

Can be used to limit which entities will go to `query_log`, say you are interested only in errors, then you can use `EXCEPTION_WHILE_PROCESSING`:

```
log_queries_min_type='EXCEPTION_WHILE_PROCESSING'
```

## log\_query\_threads

Setting up query threads logging.

Queries' threads run by ClickHouse with this setup are logged according to the rules in the `query_thread_log` server configuration parameter.

Example:

```
log_query_threads=1
```

## log\_query\_views

Setting up query views logging.

When a query run by ClickHouse with this setup on has associated views (materialized or live views), they are logged in the `query_views_log` server configuration parameter.

Example:

```
log_query_views=1
```

## log\_comment

Specifies the value for the `log_comment` field of the `system.query_log` table and comment text for the server log.

It can be used to improve the readability of server logs. Additionally, it helps to select queries related to the test from the `system.query_log` after running `clickhouse-test`.

Possible values:

- Any string no longer than `max_query_size`. If length is exceeded, the server throws an exception.

Default value: empty string.

### Example

Query:

```
SET log_comment = 'log_comment test', log_queries = 1;
SELECT 1;
SYSTEM FLUSH LOGS;
SELECT type, query FROM system.query_log WHERE log_comment = 'log_comment test' AND event_date >=
yesterday() ORDER BY event_time DESC LIMIT 2;
```

Result:

type	query
QueryStart	SELECT 1;
QueryFinish	SELECT 1;

## max\_insert\_block\_size

The size of blocks (in a count of rows) to form for insertion into a table.

This setting only applies in cases when the server forms the blocks.

For example, for an INSERT via the HTTP interface, the server parses the data format and forms blocks of the specified size.

But when using clickhouse-client, the client parses the data itself, and the 'max\_insert\_block\_size' setting on the server does not affect the size of the inserted blocks.

The setting also does not have a purpose when using INSERT SELECT, since data is inserted using the same blocks that are formed after SELECT.

Default value: 1,048,576.

The default is slightly more than `max_block_size`. The reason for this is because certain table engines (\*MergeTree) form a data part on the disk for each inserted block, which is a fairly large entity. Similarly, \*MergeTree tables sort data during insertion, and a large enough block size allow sorting more data in RAM.

## min\_insert\_block\_size\_rows

Sets the minimum number of rows in the block which can be inserted into a table by an `INSERT` query. Smaller-sized blocks are squashed into bigger ones.

Possible values:

- Positive integer.
- 0 — Squashing disabled.

Default value: 1048576.

## `min_insert_block_size_bytes`

Sets the minimum number of bytes in the block which can be inserted into a table by an `INSERT` query. Smaller-sized blocks are squashed into bigger ones.

Possible values:

- Positive integer.
- 0 — Squashing disabled.

Default value: 268435456.

## `max_replica_delay_for_distributed_queries`

Disables lagging replicas for distributed queries. See [Replication](#).

Sets the time in seconds. If a replica lags more than the set value, this replica is not used.

Default value: 300.

Used when performing `SELECT` from a distributed table that points to replicated tables.

## `max_threads`

The maximum number of query processing threads, excluding threads for retrieving data from remote servers (see the ‘`max_distributed_connections`’ parameter).

This parameter applies to threads that perform the same stages of the query processing pipeline in parallel.

For example, when reading from a table, if it is possible to evaluate expressions with functions, filter with `WHERE` and pre-aggregate for `GROUP BY` in parallel using at least ‘`max_threads`’ number of threads, then ‘`max_threads`’ are used.

Default value: the number of physical CPU cores.

For queries that are completed quickly because of a `LIMIT`, you can set a lower ‘`max_threads`’. For example, if the necessary number of entries are located in every block and `max_threads = 8`, then 8 blocks are retrieved, although it would have been enough to read just one.

The smaller the `max_threads` value, the less memory is consumed.

## `max_insert_threads`

The maximum number of threads to execute the `INSERT SELECT` query.

Possible values:

- 0 (or 1) — `INSERT SELECT` no parallel execution.
- Positive integer. Bigger than 1.

Default value: 0.

Parallel `INSERT SELECT` has effect only if the `SELECT` part is executed in parallel, see [max\\_threads](#) setting.  
Higher values will lead to higher memory usage.

## max\_compress\_block\_size

The maximum size of blocks of uncompressed data before compressing for writing to a table. By default, 1,048,576 (1 MiB). Specifying smaller block size generally leads to slightly reduced compression ratio, the compression and decompression speed increases slightly due to cache locality, and memory consumption is reduced.

### Warning

This is an expert-level setting, and you shouldn't change it if you're just getting started with ClickHouse.

Don't confuse blocks for compression (a chunk of memory consisting of bytes) with blocks for query processing (a set of rows from a table).

## min\_compress\_block\_size

For [MergeTree](#) tables. In order to reduce latency when processing queries, a block is compressed when writing the next mark if its size is at least `min_compress_block_size`. By default, 65,536.

The actual size of the block, if the uncompressed data is less than `max_compress_block_size`, is no less than this value and no less than the volume of data for one mark.

Let's look at an example. Assume that `index_granularity` was set to 8192 during table creation.

We are writing a `UInt32`-type column (4 bytes per value). When writing 8192 rows, the total will be 32 KB of data. Since `min_compress_block_size` = 65,536, a compressed block will be formed for every two marks.

We are writing a `URL` column with the `String` type (average size of 60 bytes per value). When writing 8192 rows, the average will be slightly less than 500 KB of data. Since this is more than 65,536, a compressed block will be formed for each mark. In this case, when reading data from the disk in the range of a single mark, extra data won't be decompressed.

### Warning

This is an expert-level setting, and you shouldn't change it if you're just getting started with ClickHouse.

## max\_query\_size

The maximum part of a query that can be taken to RAM for parsing with the SQL parser.  
The `INSERT` query also contains data for `INSERT` that is processed by a separate stream parser (that consumes O(1) RAM), which is not included in this restriction.

Default value: 256 KiB.

## max\_parser\_depth

Limits maximum recursion depth in the recursive descent parser. Allows controlling the stack size.

Possible values:

- Positive integer.
- 0 — Recursion depth is unlimited.

Default value: 1000.

## interactive\_delay

The interval in microseconds for checking whether request execution has been cancelled and sending the progress.

Default value: 100,000 (checks for cancelling and sends the progress ten times per second).

## connect\_timeout, receive\_timeout, send\_timeout

Timeouts in seconds on the socket used for communicating with the client.

Default value: 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.

Default value: 0

## poll\_interval

Lock in a wait loop for the specified number of seconds.

Default value: 10.

## max\_distributed\_connections

The maximum number of simultaneous connections with remote servers for distributed processing of a single query to a single Distributed table. We recommend setting a value no less than the number of servers in the cluster.

Default value: 1024.

The following parameters are only used when creating Distributed tables (and when launching a server), so there is no reason to change them at runtime.

## distributed\_connections\_pool\_size

The maximum number of simultaneous connections with remote servers for distributed processing of all queries to a single Distributed table. We recommend setting a value no less than the number of servers in the cluster.

Default value: 1024.

## max\_distributed\_depth

Limits the maximum depth of recursive queries for **Distributed** tables.

If the value is exceeded, the server throws an exception.

Possible values:

- Positive integer.
- 0 — Unlimited depth.

Default value: 5.

## **max\_replicated\_fetches\_network\_bandwidth\_for\_server**

Limits the maximum speed of data exchange over the network in bytes per second for **replicated** fetches for the server. Only has meaning at server startup. You can also limit the speed for a particular table with **max\_replicated\_fetches\_network\_bandwidth** setting.

The setting isn't followed perfectly accurately.

Possible values:

- Positive integer.
- 0 — Unlimited.

Default value: 0.

### **Usage**

Could be used for throttling speed when replicating the data to add or replace new nodes.

## **max\_replicated\_sends\_network\_bandwidth\_for\_server**

Limits the maximum speed of data exchange over the network in bytes per second for **replicated** sends for the server. Only has meaning at server startup. You can also limit the speed for a particular table with **max\_replicated\_sends\_network\_bandwidth** setting.

The setting isn't followed perfectly accurately.

Possible values:

- Positive integer.
- 0 — Unlimited.

Default value: 0.

### **Usage**

Could be used for throttling speed when replicating the data to add or replace new nodes.

## **connect\_timeout\_with\_failover\_ms**

The timeout in milliseconds for connecting to a remote server for a Distributed table engine, if the 'shard' and 'replica' sections are used in the cluster definition.

If unsuccessful, several attempts are made to connect to various replicas.

Default value: 50.

## **connection\_pool\_max\_wait\_ms**

The wait time in milliseconds for a connection when the connection pool is full.

Possible values:

- Positive integer.

- 0 — Infinite timeout.

Default value: 0.

## connections\_with\_failover\_max\_tries

The maximum number of connection attempts with each replica for the Distributed table engine.

Default value: 3.

## extremes

Whether to count extreme values (the minimums and maximums in columns of a query result). Accepts 0 or 1. By default, 0 (disabled).

For more information, see the section “Extreme values”.

## kafka\_max\_wait\_ms

The wait time in milliseconds for reading messages from [Kafka](#) before retry.

Possible values:

- Positive integer.
- 0 — Infinite timeout.

Default value: 5000.

See also:

- [Apache Kafka](#)

## use\_uncompressed\_cache

Whether to use a cache of uncompressed blocks. Accepts 0 or 1. By default, 0 (disabled).

Using the uncompressed cache (only for tables in the MergeTree family) can significantly reduce latency and increase throughput when working with a large number of short queries. Enable this setting for users who send frequent short requests. Also pay attention to the [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.

For queries that read at least a somewhat large volume of data (one million rows or more), the uncompressed cache is disabled automatically to save space for truly small queries. This means that you can keep the ‘use\_uncompressed\_cache’ setting always set to 1.

## replace\_running\_query

When using the HTTP interface, the ‘query\_id’ parameter can be passed. This is any string that serves as the query identifier.

If a query from the same user with the same ‘query\_id’ already exists at this time, the behaviour depends on the ‘replace\_running\_query’ parameter.

0 (default) – Throw an exception (do not allow the query to run if a query with the same ‘query\_id’ is already running).

1 – Cancel the old query and start running the new one.

Yandex.Metrica uses this parameter set to 1 for implementing suggestions for segmentation conditions. After entering the next character, if the old query hasn’t finished yet, it should be cancelled.

## `replace_running_query_max_wait_ms`

The wait time for running the query with the same `query_id` to finish, when the [replace\\_running\\_query](#) setting is active.

Possible values:

- Positive integer.
- 0 — Throwing an exception that does not allow to run a new query if the server already executes a query with the same `query_id`.

Default value: 5000.

## `stream_flush_interval_ms`

Works for tables with streaming in the case of a timeout, or when a thread generates [max\\_insert\\_block\\_size](#) rows.

The default value is 7500.

The smaller the value, the more often data is flushed into the table. Setting the value too low leads to poor performance.

## `load_balancing`

Specifies the algorithm of replicas selection that is used for distributed query processing.

ClickHouse supports the following algorithms of choosing replicas:

- [Random](#) (by default)
- [Nearest hostname](#)
- [In order](#)
- [First or random](#)
- [Round robin](#)

See also:

- [distributed\\_replica\\_max\\_ignored\\_errors](#)

### [Random \(by Default\)](#)

```
load_balancing = random
```

The number of errors is counted for each replica. The query is sent to the replica with the fewest errors, and if there are several of these, to anyone of them.

Disadvantages: Server proximity is not accounted for; if the replicas have different data, you will also get different data.

### [Nearest Hostname](#)

```
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).

For instance, `example01-01-1` and `example01-01-2.yandex.ru` are different in one position, while `example01-01-1` and `example01-02-2` differ in two places.

This method might seem primitive, but it does not require external data about network topology, and it does not compare IP addresses, which would be complicated for our IPv6 addresses.

Thus, if there are equivalent replicas, the closest one by name is preferred.

We can also assume that when sending a query to the same server, in the absence of failures, a distributed query will also go to the same servers. So even if different data is placed on the replicas, the query will return mostly the same results.

## In Order

```
load_balancing = in_order
```

Replicas with the same number of errors are accessed in the same order as they are specified in the configuration.

This method is appropriate when you know exactly which replica is preferable.

## First or Random

```
load_balancing = first_or_random
```

This algorithm chooses the first replica in the set or a random replica if the first is unavailable. It's effective in cross-replication topology setups, but useless in other configurations.

The `first_or_random` algorithm solves the problem of the `in_order` algorithm. With `in_order`, if one replica goes down, the next one gets a double load while the remaining replicas handle the usual amount of traffic. When using the `first_or_random` algorithm, the load is evenly distributed among replicas that are still available.

It's possible to explicitly define what the first replica is by using the setting `load_balancing_first_offset`. This gives more control to rebalance query workloads among replicas.

## Round Robin

```
load_balancing = round_robin
```

This algorithm uses a round-robin policy across replicas with the same number of errors (only the queries with `round_robin` policy is accounted).

## `prefer_localhost_replica`

Enables/disables preferable using the localhost replica when processing distributed queries.

Possible values:

- 1 — ClickHouse always sends a query to the localhost replica if it exists.
- 0 — ClickHouse uses the balancing strategy specified by the `load_balancing` setting.

Default value: 1.

## Warning

Disable this setting if you use **max\_parallel\_replicas**.

## totals\_mode

How to calculate TOTALS when HAVING is present, as well as when max\_rows\_to\_group\_by and group\_by\_overflow\_mode = 'any' are present.

See the section "WITH TOTALS modifier".

## totals\_auto\_threshold

The threshold for totals\_mode = 'auto'.

See the section "WITH TOTALS modifier".

## max\_parallel\_replicas

The maximum number of replicas for each shard when executing a query.

Possible values:

- Positive integer.

Default value: 1.

### Additional Info

This setting is useful for replicated tables with a sampling key. A query may be processed faster if it is executed on several servers in parallel. But the query performance may degrade in the following cases:

- The position of the sampling key in the partitioning key does not allow efficient range scans.
- Adding a sampling key to the table makes filtering by other columns less efficient.
- The sampling key is an expression that is expensive to calculate.
- The cluster latency distribution has a long tail, so that querying more servers increases the query overall latency.

## Warning

This setting will produce incorrect results when joins or subqueries are involved, and all tables don't meet certain requirements. See **Distributed Subqueries and max\_parallel\_replicas** for more details.

## compile\_expressions

Enables or disables compilation of frequently used simple functions and operators to native code with LLVM at runtime.

Possible values:

- 0 — Disabled.

- 1 — Enabled.

Default value: 1.

## min\_count\_to\_compile\_expression

Minimum count of executing same expression before it is get compiled.

Default value: 3.

## output\_format\_json\_quote\_64bit\_integers

Controls quoting of 64-bit or bigger integers (like UInt64 or Int128) when they are output in a JSON format. Such integers are enclosed in quotes by default. This behavior is compatible with most JavaScript implementations.

Possible values:

- 0 — Integers are output without quotes.
- 1 — Integers are enclosed in quotes.

Default value: 1.

## output\_format\_json\_quote\_denormals

Enables +nan, -nan, +inf, -inf outputs in JSON output format.

Possible values:

- 0 — Disabled.
- 1 — Enabled.

Default value: 0.

### Example

Consider the following table account\_orders:

id	name	duration	period	area
1	Andrew	20	0	400
2	John	40	0	0
3	Bob	15	0	-100

When output\_format\_json\_quote\_denormals = 0, the query returns null values in output:

```
SELECT area/period FROM account_orders FORMAT JSON;
```

```
{
  "meta": [
    {
      "name": "divide(area, period)",
      "type": "Float64"
    }
  ],
  "data": [
    {
      "divide(area, period)": null
    },
    {
      "divide(area, period)": null
    },
    {
      "divide(area, period)": null
    }
  ],
  "rows": 3,
  "statistics": {
    "elapsed": 0.003648093,
    "rows_read": 3,
    "bytes_read": 24
  }
}
```

When `output_format_json_quote_denormals = 1`, the query returns:

```
{
  "meta": [
    {
      "name": "divide(area, period)",
      "type": "Float64"
    }
  ],
  "data": [
    {
      "divide(area, period)": "inf"
    },
    {
      "divide(area, period)": "-nan"
    },
    {
      "divide(area, period)": "-inf"
    }
  ],
  "rows": 3,
  "statistics": {
    "elapsed": 0.000070241,
    "rows_read": 3,
    "bytes_read": 24
  }
}
```

## format\_csv\_delimiter

The character is interpreted as a delimiter in the CSV data. By default, the delimiter is `,`.

## `input_format_csv_unquoted_null_literal_as_null`

For CSV input format enables or disables parsing of unquoted `NULL` as literal (synonym for `\N`).

## `input_format_csv_enum_as_number`

Enables or disables parsing enum values as enum ids for CSV input format.

Possible values:

- 0 — Enum values are parsed as values.
- 1 — Enum values are parsed as enum IDs.

Default value: 0.

### **Examples**

Consider the table:

```
CREATE TABLE table_with_enum_column_for_csv_insert (Id Int32,Value Enum('first' = 1, 'second' = 2))
ENGINE=Memory();
```

When the `input_format_csv_enum_as_number` setting is enabled:

```
SET input_format_csv_enum_as_number = 1;
INSERT INTO table_with_enum_column_for_csv_insert FORMAT CSV 102,2;
SELECT * FROM table_with_enum_column_for_csv_insert;
```

Result:

Id	Value
102	second

When the `input_format_csv_enum_as_number` setting is disabled, the `INSERT` query:

```
SET input_format_csv_enum_as_number = 0;
INSERT INTO table_with_enum_column_for_csv_insert FORMAT CSV 102,2;
```

throws an exception.

## `output_format_csv_crlf_end_of_line`

Use DOS/Windows-style line separator (CRLF) in CSV instead of Unix style (LF).

## `output_format_tsv_crlf_end_of_line`

Use DOS/Windows-style line separator (CRLF) in TSV instead of Unix style (LF).

## `insert_quorum`

Enables the quorum writes.

- If `insert_quorum < 2`, the quorum writes are disabled.
- If `insert_quorum >= 2`, the quorum writes are enabled.

Default value: 0.

#### Quorum writes

`INSERT` succeeds only when ClickHouse manages to correctly write data to the `insert_quorum` of replicas during the `insert_quorum_timeout`. If for any reason the number of replicas with successful writes does not reach the `insert_quorum`, the write is considered failed and ClickHouse will delete the inserted block from all the replicas where data has already been written.

All the replicas in the quorum are consistent, i.e., they contain data from all previous `INSERT` queries. The `INSERT` sequence is linearized.

When reading the data written from the `insert_quorum`, you can use the `select_sequential_consistency` option.

ClickHouse generates an exception

- If the number of available replicas at the time of the query is less than the `insert_quorum`.
- At an attempt to write data when the previous block has not yet been inserted in the `insert_quorum` of replicas. This situation may occur if the user tries to perform an `INSERT` before the previous one with the `insert_quorum` is completed.

See also:

- [insert\\_quorum\\_timeout](#)
- [select\\_sequential\\_consistency](#)

## insert\_quorum\_timeout

Write to a quorum timeout in milliseconds. If the timeout has passed and no write has taken place yet, ClickHouse will generate an exception and the client must repeat the query to write the same block to the same or any other replica.

Default value: 600 000 milliseconds (ten minutes).

See also:

- [insert\\_quorum](#)
- [select\\_sequential\\_consistency](#)

## select\_sequential\_consistency

Enables or disables sequential consistency for `SELECT` queries:

Possible values:

- 0 — Disabled.
- 1 — Enabled.

Default value: 0.

#### Usage

When sequential consistency is enabled, ClickHouse allows the client to execute the `SELECT` query only for those replicas that contain data from all previous `INSERT` queries executed with `insert_quorum`. If the client refers to a partial replica, ClickHouse will generate an exception. The `SELECT` query will not include data that has not yet been written to the quorum of replicas.

See also:

- [insert\\_quorum](#)
- [insert\\_quorum\\_timeout](#)

## insert\_deduplicate

Enables or disables block deduplication of `INSERT` (for Replicated\* tables).

Possible values:

- 0 — Disabled.
- 1 — Enabled.

Default value: 1.

By default, blocks inserted into replicated tables by the `INSERT` statement are deduplicated (see [Data Replication](#)).

## deduplicate\_blocks\_in\_dependent\_materialized\_views

Enables or disables the deduplication check for materialized views that receive data from Replicated\* tables.

Possible values:

- 0 — Disabled.
  - 1 — Enabled.

Default value: 0.

### Usage

By default, deduplication is not performed for materialized views but is done upstream, in the source table. If an `INSERT` block is skipped due to deduplication in the source table, there will be no insertion into attached materialized views. This behaviour exists to enable the insertion of highly aggregated data into materialized views, for cases where inserted blocks are the same after materialized view aggregation but derived from different `INSERT`s into the source table.

At the same time, this behaviour “breaks” `INSERT` idempotency. If an `INSERT` into the main table was successful and `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` allows for changing this behaviour. On retry, a materialized view will receive the repeat insert and will perform a deduplication check by itself, ignoring check result for the source table, and will insert rows lost because of the first failure.

## max\_network\_bytes

Limits the data volume (in bytes) that is received or transmitted over the network when executing a query. This setting applies to every individual query.

Possible values:

- Positive integer.
- 0 — Data volume control is disabled.

Default value: 0.

## max\_network\_bandwidth

Limits the speed of the data exchange over the network in bytes per second. This setting applies to every query.

Possible values:

- Positive integer.
- 0 — Bandwidth control is disabled.

Default value: 0.

## max\_network\_bandwidth\_for\_user

Limits the speed of the data exchange over the network in bytes per second. This setting applies to all concurrently running queries performed by a single user.

Possible values:

- Positive integer.
- 0 — Control of the data speed is disabled.

Default value: 0.

## max\_network\_bandwidth\_for\_all\_users

Limits the speed that data is exchanged at over the network in bytes per second. This setting applies to all concurrently running queries on the server.

Possible values:

- Positive integer.
- 0 — Control of the data speed is disabled.

Default value: 0.

## count\_distinctImplementation

Specifies which of the uniq\* functions should be used to perform the `COUNT(DISTINCT ...)` construction.

Possible values:

- `uniq`
- `uniqCombined`
- `uniqCombined64`
- `uniqHLL12`
- `uniqExact`

Default value: `uniqExact`.

## skip\_unavailable\_shards

Enables or disables silently skipping of unavailable shards.

Shard is considered unavailable if all its replicas are unavailable. A replica is unavailable in the following cases:

- ClickHouse can't connect to replica for any reason.

When connecting to a replica, ClickHouse performs several attempts. If all these attempts fail, the replica is considered unavailable.

- Replica can't be resolved through DNS.

If replica's hostname can't be resolved through DNS, it can indicate the following situations:

- Replica's host has no DNS record. It can occur in systems with dynamic DNS, for example, [Kubernetes](#), where nodes can be unresolvable during downtime, and this is not an error.
- Configuration error. ClickHouse configuration file contains a wrong hostname.

Possible values:

- 1 — skipping enabled.

If a shard is unavailable, ClickHouse returns a result based on partial data and does not report node availability issues.

- 0 — skipping disabled.

If a shard is unavailable, ClickHouse throws an exception.

Default value: 0.

## distributed\_group\_by\_no\_merge

Do not merge aggregation states from different servers for distributed query processing, you can use this in case it is for certain that there are different keys on different shards

Possible values:

- 0 — Disabled (final query processing is done on the initiator node).
- 1 - Do not merge aggregation states from different servers for distributed query processing (query completely processed on the shard, initiator only proxy the data), can be used in case it is for certain that there are different keys on different shards.
- 2 - Same as 1 but applies ORDER BY and LIMIT (it is not possible when the query processed completely on the remote node, like for `distributed_group_by_no_merge=1`) on the initiator (can be used for queries with ORDER BY and/or LIMIT).

Default value: 0

### Example

```
SELECT *
FROM remote('127.0.0.{2,3}', system.one)
GROUP BY dummy
LIMIT 1
SETTINGS distributed_group_by_no_merge = 1
FORMAT PrettyCompactMonoBlock
```

dummy	0	0
-------	---	---

```
SELECT *
FROM remote('127.0.0.{2,3}', system.one)
GROUP BY dummy
LIMIT 1
SETTINGS distributed_group_by_no_merge = 2
FORMAT PrettyCompactMonoBlock
```

```
└── dummy └
    └── 0 |
```

## distributed\_push\_down\_limit

Enables or disables **LIMIT** applying on each shard separately.

This will allow to avoid:

- Sending extra rows over network;
- Processing rows behind the limit on the initiator.

Starting from 21.9 version you cannot get inaccurate results anymore, since `distributed_push_down_limit` changes query execution only if at least one of the conditions met:

- `distributed_group_by_no_merge > 0`.
- Query **does not have** GROUP BY/DISTINCT/LIMIT BY, but it has ORDER BY/LIMIT.
- Query **has** GROUP BY/DISTINCT/LIMIT BY with ORDER BY/LIMIT and:
  - `optimize_skip_unused_shards` is enabled.
  - `optimize_distributed_group_by_sharding_key` is enabled.

Possible values:

- 0 — Disabled.
- 1 — Enabled.

Default value: 1.

See also:

- [distributed\\_group\\_by\\_no\\_merge](#)
- [optimize\\_skip\\_unused\\_shards](#)
- [optimize\\_distributed\\_group\\_by\\_sharding\\_key](#)

## optimize\_skip\_unused\_shards\_limit

Limit for number of sharding key values, turns off `optimize_skip_unused_shards` if the limit is reached.

Too many values may require significant amount for processing, while the benefit is doubtful, since if you have huge number of values in IN (...), then most likely the query will be sent to all shards anyway.

Default value: 1000

## optimize\_skip\_unused\_shards

Enables or disables skipping of unused shards for **SELECT** queries that have sharding key condition in **WHERE/PREWHERE** (assuming that the data is distributed by sharding key, otherwise a query yields incorrect result).

Possible values:

- 0 — Disabled.

- 1 — Enabled.

Default value: 0

## optimize\_skip\_unused\_shards\_rewrite\_in

Rewrite IN in query for remote shards to exclude values that does not belong to the shard (requires `optimize_skip_unused_shards`).

Possible values:

- 0 — Disabled.
- 1 — Enabled.

Default value: 1 (since it requires `optimize_skip_unused_shards` anyway, which 0 by default)

## allow\_nondeterministic\_optimize\_skip\_unused\_shards

Allow nondeterministic (like `rand` or `dictGet`, since later has some caveats with updates) functions in sharding key.

Possible values:

- 0 — Disallowed.
- 1 — Allowed.

Default value: 0

## optimize\_skip\_unused\_shards\_nesting

Controls `optimize_skip_unused_shards` (hence still requires `optimize_skip_unused_shards`) depends on the nesting level of the distributed query (case when you have `Distributed` table that look into another `Distributed` table).

Possible values:

- 0 — Disabled, `optimize_skip_unused_shards` works always.
- 1 — Enables `optimize_skip_unused_shards` only for the first level.
- 2 — Enables `optimize_skip_unused_shards` up to the second level.

Default value: 0

## force\_optimize\_skip\_unused\_shards

Enables or disables query execution if `optimize_skip_unused_shards` is enabled and skipping of unused shards is not possible. If the skipping is not possible and the setting is enabled, an exception will be thrown.

Possible values:

- 0 — Disabled. ClickHouse does not throw an exception.
- 1 — Enabled. Query execution is disabled only if the table has a sharding key.
- 2 — Enabled. Query execution is disabled regardless of whether a sharding key is defined for the table.

Default value: 0

## force\_optimize\_skip\_unused\_shards\_nesting

Controls `force_optimize_skip_unused_shards` (hence still requires `force_optimize_skip_unused_shards`) depends on the nesting level of the distributed query (case when you have `Distributed` table that look into another `Distributed` table).

Possible values:

- 0 - Disabled, `force_optimize_skip_unused_shards` works always.
- 1 — Enables `force_optimize_skip_unused_shards` only for the first level.
- 2 — Enables `force_optimize_skip_unused_shards` up to the second level.

Default value: 0

## `optimize_distributed_group_by_sharding_key`

Optimize GROUP BY sharding\_key queries, by avoiding costly aggregation on the initiator server (which will reduce memory usage for the query on the initiator server).

The following types of queries are supported (and all combinations of them):

- `SELECT DISTINCT [..., ]sharding_key[, ...] FROM dist`
- `SELECT ... FROM dist GROUP BY sharding_key[, ...]`
- `SELECT ... FROM dist GROUP BY sharding_key[, ...] ORDER BY x`
- `SELECT ... FROM dist GROUP BY sharding_key[, ...] LIMIT 1`
- `SELECT ... FROM dist GROUP BY sharding_key[, ...] LIMIT 1 BY x`

The following types of queries are not supported (support for some of them may be added later):

- `SELECT ... GROUP BY sharding_key[, ...] WITH TOTALS`
- `SELECT ... GROUP BY sharding_key[, ...] WITH ROLLUP`
- `SELECT ... GROUP BY sharding_key[, ...] WITH CUBE`
- `SELECT ... GROUP BY sharding_key[, ...] SETTINGS extremes=1`

Possible values:

- 0 — Disabled.
- 1 — Enabled.

Default value: 0

See also:

- [distributed\\_group\\_by\\_no\\_merge](#)
- [distributed\\_push\\_down\\_limit](#)
- [optimize\\_skip\\_unused\\_shards](#)

### Note

Right now it requires `optimize_skip_unused_shards` (the reason behind this is that one day it may be enabled by default, and it will work correctly only if data was inserted via `Distributed` table, i.e. data is distributed according to `sharding_key`).

## `optimize_throw_if_noop`

Enables or disables throwing an exception if an `OPTIMIZE` query didn't perform a merge.

By default, `OPTIMIZE` returns successfully even if it didn't do anything. This setting lets you differentiate these situations and get the reason in an exception message.

Possible values:

- 1 — Throwing an exception is enabled.
- 0 — Throwing an exception is disabled.

Default value: 0.

## `optimize_functions_to_subcolumns`

Enables or disables optimization by transforming some functions to reading subcolumns. This reduces the amount of data to read.

These functions can be transformed:

- `length` to read the `size0` subcolumn.
- `empty` to read the `size0` subcolumn.
- `notEmpty` to read the `size0` subcolumn.
- `isNull` to read the `null` subcolumn.
- `isNotNull` to read the `null` subcolumn.
- `count` to read the `null` subcolumn.
- `mapKeys` to read the `keys` subcolumn.
- `mapValues` to read the `values` subcolumn.

Possible values:

- 0 — Optimization disabled.
- 1 — Optimization enabled.

Default value: 0.

## `optimize_trivial_count_query`

Enables or disables the optimization to trivial query `SELECT count() FROM table` using metadata from MergeTree. If you need to use row-level security, disable this setting.

Possible values:

- 0 — Optimization disabled.
- 1 — Optimization enabled.

Default value: 1.

See also:

- [optimize\\_functions\\_to\\_subcolumns](#)

## `distributed_replica_error_half_life`

- Type: seconds
- Default value: 60 seconds

Controls how fast errors in distributed tables are zeroed. If a replica is unavailable for some time, accumulates 5 errors, and `distributed_replica_error_half_life` is set to 1 second, then the replica is considered normal 3 seconds after the last error.

See also:

- [load\\_balancing](#)
- [Table engine Distributed](#)
- [distributed\\_replica\\_error\\_cap](#)
- [distributed\\_replica\\_max\\_ignored\\_errors](#)

## `distributed_replica_error_cap`

- Type: unsigned int
- Default value: 1000

The error count of each replica is capped at this value, preventing a single replica from accumulating too many errors.

See also:

- [load\\_balancing](#)
- [Table engine Distributed](#)
- [distributed\\_replica\\_error\\_half\\_life](#)
- [distributed\\_replica\\_max\\_ignored\\_errors](#)

## `distributed_replica_max_ignored_errors`

- Type: unsigned int
- Default value: 0

The number of errors that will be ignored while choosing replicas (according to `load_balancing` algorithm).

See also:

- [load\\_balancing](#)
- [Table engine Distributed](#)
- [distributed\\_replica\\_error\\_cap](#)
- [distributed\\_replica\\_error\\_half\\_life](#)

## `distributed_directory_monitor_sleep_time_ms`

Base interval for the [Distributed](#) table engine to send data. The actual interval grows exponentially in the event of errors.

Possible values:

- A positive integer number of milliseconds.

Default value: 100 milliseconds.

## distributed\_directory\_monitor\_max\_sleep\_time\_ms

Maximum interval for the **Distributed** table engine to send data. Limits exponential growth of the interval set in the [distributed\\_directory\\_monitor\\_sleep\\_time\\_ms](#) setting.

Possible values:

- A positive integer number of milliseconds.

Default value: 30000 milliseconds (30 seconds).

## distributed\_directory\_monitor\_batch\_inserts

Enables/disables inserted data sending in batches.

When batch sending is enabled, the **Distributed** table engine tries to send multiple files of inserted data in one operation instead of sending them separately. Batch sending improves cluster performance by better utilizing server and network resources.

Possible values:

- 1 — Enabled.
- 0 — Disabled.

Default value: 0.

## distributed\_directory\_monitor\_split\_batch\_on\_failure

Enables/disables splitting batches on failures.

Sometimes sending particular batch to the remote shard may fail, because of some complex pipeline after (i.e. `MATERIALIZED VIEW` with `GROUP BY`) due to Memory limit exceeded or similar errors. In this case, retrying will not help (and this will stuck distributed sends for the table) but sending files from that batch one by one may succeed `INSERT`.

So installing this setting to `1` will disable batching for such batches (i.e. temporary disables `distributed_directory_monitor_batch_inserts` for failed batches).

Possible values:

- 1 — Enabled.
- 0 — Disabled.

Default value: 0.

### Note

This setting also affects broken batches (that may appear because of abnormal server (machine) termination and `no fsync_after_insert/fsync_directories` for **Distributed** table engine).

### Warning

You should not rely on automatic batch splitting, since this may hurt performance.

## os\_thread\_priority

Sets the priority ([nice](#)) for threads that execute queries. The OS scheduler considers this priority when choosing the next thread to run on each available CPU core.

### Warning

To use this setting, you need to set the `CAP_SYS_NICE` capability. The `clickhouse-server` package sets it up during installation. Some virtual environments do not allow you to set the `CAP_SYS_NICE` capability. In this case, `clickhouse-server` shows a message about it at the start.

Possible values:

- You can set values in the range [-20, 19].

Lower values mean higher priority. Threads with low `nice` priority values are executed more frequently than threads with high values. High values are preferable for long-running non-interactive queries because it allows them to quickly give up resources in favour of short interactive queries when they arrive.

Default value: 0.

## query\_profiler\_real\_time\_period\_ns

Sets the period for a real clock timer of the [query profiler](#). Real clock timer counts wall-clock time.

Possible values:

- Positive integer number, in nanoseconds.

Recommended values:

- 10000000 (100 times a second) nanoseconds and less for single queries.
- 1000000000 (once a second) for cluster-wide profiling.

- 0 for turning off the timer.

Type: [UInt64](#).

Default value: 1000000000 nanoseconds (once a second).

See also:

- System table [trace\\_log](#)

## query\_profiler\_cpu\_time\_period\_ns

Sets the period for a CPU clock timer of the [query profiler](#). This timer counts only CPU time.

Possible values:

- A positive integer number of nanoseconds.

Recommended values:

- 10000000 (100 times a second) nanoseconds and more for single queries.
  - 1000000000 (once a second) for cluster-wide profiling.

- 0 for turning off the timer.

Type: [UInt64](#).

Default value: 1000000000 nanoseconds.

See also:

- System table [trace\\_log](#)

## allow\_introspection\_functions

Enables or disables [introspections functions](#) for query profiling.

Possible values:

- 1 — Introspection functions enabled.
- 0 — Introspection functions disabled.

Default value: 0.

### See Also

- [Sampling Query Profiler](#)
- System table [trace\\_log](#)

## input\_format\_parallel\_parsing

Enables or disables order-preserving parallel parsing of data formats. Supported only for [TSV](#), [TKSV](#), [CSV](#) and [JSONEachRow](#) formats.

Possible values:

- 1 — Enabled.
- 0 — Disabled.

Default value: 0.

## output\_format\_parallel\_formatting

Enables or disables parallel formatting of data formats. Supported only for [TSV](#), [TKSV](#), [CSV](#) and [JSONEachRow](#) formats.

Possible values:

- 1 — Enabled.
- 0 — Disabled.

Default value: 0.

## min\_chunk\_bytes\_for\_parallel\_parsing

- Type: unsigned int
- Default value: 1 MiB

The minimum chunk size in bytes, which each thread will parse in parallel.

## output\_format\_avro\_codec

Sets the compression codec used for output Avro file.

Type: string

Possible values:

- `null` — No compression
- `deflate` — Compress with Deflate (zlib)
- `snappy` — Compress with [Snappy](#)

Default value: `snappy` (if available) or `deflate`.

## output\_format\_avro\_sync\_interval

Sets minimum data size (in bytes) between synchronization markers for output Avro file.

Type: unsigned int

Possible values: 32 (32 bytes) - 1073741824 (1 GiB)

Default value: 32768 (32 KiB)

## output\_format\_avro\_string\_column\_pattern

Regexp of column names of type String to output as Avro `string` (default is `bytes`).  
RE2 syntax is supported.

Type: string

## format\_avro\_schema\_registry\_url

Sets [Confluent Schema Registry](#) URL to use with [AvroConfluent](#) format.

Default value: `Empty`.

## input\_format\_avro\_allow\_missing\_fields

Enables using fields that are not specified in [Avro](#) or [AvroConfluent](#) format schema. When a field is not found in the schema, ClickHouse uses the default value instead of throwing an exception.

Possible values:

- 0 — Disabled.
- 1 — Enabled.

Default value: 0.

## background\_pool\_size

Sets the number of threads performing background operations in table engines (for example, merges in [MergeTree engine](#) tables). This setting is applied from the `default` profile at the ClickHouse server start and can't be changed in a user session. By adjusting this setting, you manage CPU and disk load. Smaller pool size utilizes less CPU and disk resources, but background processes advance slower which might eventually impact query performance.

Before changing it, please also take a look at related [MergeTree settings](#), such as `number_of_free_entries_in_pool_to_lower_max_size_of_merge` and `number_of_free_entries_in_pool_to_execute_mutation`.

Possible values:

- Any positive integer.

Default value: 16.

## merge\_selecting\_sleep\_ms

Sleep time for merge selecting when no part is selected. A lower setting triggers selecting tasks in `background_schedule_pool` frequently, which results in a large number of requests to Zookeeper in large-scale clusters.

Possible values:

- Any positive integer.

Default value: 5000.

## parallel\_distributed\_insert\_select

Enables parallel distributed `INSERT ... SELECT` query.

If we execute `INSERT INTO distributed_table_a SELECT ... FROM distributed_table_b` queries and both tables use the same cluster, and both tables are either [replicated](#) or non-replicated, then this query is processed locally on every shard.

Possible values:

- 0 — Disabled.
- 1 — `SELECT` will be executed on each shard from the underlying table of the distributed engine.
- 2 — `SELECT` and `INSERT` will be executed on each shard from/to the underlying table of the distributed engine.

Default value: 0.

## insert\_distributed\_sync

Enables or disables synchronous data insertion into a [Distributed](#) table.

By default, when inserting data into a `Distributed` table, the ClickHouse server sends data to cluster nodes in asynchronous mode. When `insert_distributed_sync=1`, the data is processed synchronously, and the `INSERT` operation succeeds only after all the data is saved on all shards (at least one replica for each shard if `internal_replication` is true).

Possible values:

- 0 — Data is inserted in asynchronous mode.
- 1 — Data is inserted in synchronous mode.

Default value: 0.

## See Also

- [Distributed Table Engine](#)
- [Managing Distributed Tables](#)

## insert\_distributed\_one\_random\_shard

Enables or disables random shard insertion into a [Distributed](#) table when there is no distributed key.

By default, when inserting data into a [Distributed](#) table with more than one shard, the ClickHouse server will reject any insertion request if there is no distributed key. When `insert_distributed_one_random_shard = 1`, insertions are allowed and data is forwarded randomly among all shards.

Possible values:

- 0 — Insertion is rejected if there are multiple shards and no distributed key is given.
- 1 — Insertion is done randomly among all available shards when no distributed key is given.

Default value: 0.

## insert\_shard\_id

If not 0, specifies the shard of [Distributed](#) table into which the data will be inserted synchronously.

If `insert_shard_id` value is incorrect, the server will throw an exception.

To get the number of shards on `requested_cluster`, you can check server config or use this query:

```
SELECT uniq(shard_num) FROM system.clusters WHERE cluster = 'requested_cluster';
```

Possible values:

- 0 — Disabled.
- Any number from 1 to `shards_num` of corresponding [Distributed](#) table.

Default value: 0.

## Example

Query:

```
CREATE TABLE x AS system.numbers ENGINE = MergeTree ORDER BY number;
CREATE TABLE x_dist AS x ENGINE = Distributed('test_cluster_two_shards_localhost', currentDatabase(), x);
INSERT INTO x_dist SELECT * FROM numbers(5) SETTINGS insert_shard_id = 1;
SELECT * FROM x_dist ORDER BY number ASC;
```

Result:

number
0
0
1
1
2
2
3
3
4
4

## use\_compact\_format\_in\_distributed\_parts\_names

Uses compact format for storing blocks for async (`insert_distributed_sync`) INSERT into tables with `Distributed` engine.

Possible values:

- 0 — Uses `user[:password]@host:port#default_database` directory format.
- 1 — Uses `[shard{shard_index}][_replica{replica_index}]` directory format.

Default value: 1.

### Note

- with `use_compact_format_in_distributed_parts_names=0` changes from cluster definition will not be applied for async INSERT.
- with `use_compact_format_in_distributed_parts_names=1` changing the order of the nodes in the cluster definition, will change the `shard_index/replica_index` so be aware.

## background\_buffer\_flush\_schedule\_pool\_size

Sets the number of threads performing background flush in `Buffer`-engine tables. This setting is applied at the ClickHouse server start and can't be changed in a user session.

Possible values:

- Any positive integer.

Default value: 16.

## background\_move\_pool\_size

Sets the number of threads performing background moves of data parts for `MergeTree`-engine tables. This setting is applied at the ClickHouse server start and can't be changed in a user session.

Possible values:

- Any positive integer.

Default value: 8.

## background\_schedule\_pool\_size

Sets the number of threads performing background tasks for `replicated` tables, `Kafka` streaming, `DNS cache updates`. This setting is applied at ClickHouse server start and can't be changed in a user session.

Possible values:

- Any positive integer.

Default value: 128.

## background\_fetches\_pool\_size

Sets the number of threads performing background fetches for **replicated** tables. This setting is applied at the ClickHouse server start and can't be changed in a user session. For production usage with frequent small insertions or slow ZooKeeper cluster is recommended to use default value.

Possible values:

- Any positive integer.

Default value: 8.

## always\_fetch\_merged\_part

Prohibits data parts merging in **Replicated\*MergeTree**-engine tables.

When merging is prohibited, the replica never merges parts and always downloads merged parts from other replicas. If there is no required data yet, the replica waits for it. CPU and disk load on the replica server decreases, but the network load on the cluster increases. This setting can be useful on servers with relatively weak CPUs or slow disks, such as servers for backups storage.

Possible values:

- 0 — **Replicated\*MergeTree**-engine tables merge data parts at the replica.
- 1 — **Replicated\*MergeTree**-engine tables do not merge data parts at the replica. The tables download merged data parts from other replicas.

Default value: 0.

### See Also

- [Data Replication](#)

## background\_distributed\_schedule\_pool\_size

Sets the number of threads performing background tasks for **distributed** sends. This setting is applied at the ClickHouse server start and can't be changed in a user session.

Possible values:

- Any positive integer.

Default value: 16.

## background\_message\_broker\_schedule\_pool\_size

Sets the number of threads performing background tasks for message streaming. This setting is applied at the ClickHouse server start and can't be changed in a user session.

Possible values:

- Any positive integer.

Default value: 16.

## See Also

- [Kafka](#) engine.
- [RabbitMQ](#) engine.

## validate\_polygons

Enables or disables throwing an exception in the `pointInPolygon` function, if the polygon is self-intersecting or self-tangent.

Possible values:

- 0 — Throwing an exception is disabled. `pointInPolygon` accepts invalid polygons and returns possibly incorrect results for them.
- 1 — Throwing an exception is enabled.

Default value: 1.

## transform\_null\_in

Enables equality of `NULL` values for `IN` operator.

By default, `NULL` values can't be compared because `NULL` means undefined value. Thus, comparison `expr = NULL` must always return `false`. With this setting `NULL = NULL` returns `true` for `IN` operator.

Possible values:

- 0 — Comparison of `NULL` values in `IN` operator returns `false`.
- 1 — Comparison of `NULL` values in `IN` operator returns `true`.

Default value: 0.

### Example

Consider the `null_in` table:

idx	i
1	1
2	NULL
3	3

Query:

```
SELECT idx, i FROM null_in WHERE i IN (1, NULL) SETTINGS transform_null_in = 0;
```

Result:

idx	i
1	1

Query:

```
SELECT idx, i FROM null_in WHERE i IN (1, NULL) SETTINGS transform_null_in = 1;
```

Result:

idx	i
1	1
2	NULL

## See Also

- [NULL Processing in IN Operators](#)

## low\_cardinality\_max\_dictionary\_size

Sets a maximum size in rows of a shared global dictionary for the [LowCardinality](#) data type that can be written to a storage file system. This setting prevents issues with RAM in case of unlimited dictionary growth. All the data that can't be encoded due to maximum dictionary size limitation ClickHouse writes in an ordinary method.

Possible values:

- Any positive integer.

Default value: 8192.

## low\_cardinality\_use\_single\_dictionary\_for\_part

Turns on or turns off using of single dictionary for the data part.

By default, the ClickHouse server monitors the size of dictionaries and if a dictionary overflows then the server starts to write the next one. To prohibit creating several dictionaries set `low_cardinality_use_single_dictionary_for_part = 1`.

Possible values:

- 1 — Creating several dictionaries for the data part is prohibited.
- 0 — Creating several dictionaries for the data part is not prohibited.

Default value: 0.

## low\_cardinality\_allow\_in\_native\_format

Allows or restricts using the [LowCardinality](#) data type with the [Native](#) format.

If usage of `LowCardinality` is restricted, ClickHouse server converts `LowCardinality`-columns to ordinary ones for `SELECT` queries, and convert ordinary columns to `LowCardinality`-columns for `INSERT` queries.

This setting is required mainly for third-party clients which do not support `LowCardinality` data type.

Possible values:

- 1 — Usage of `LowCardinality` is not restricted.
- 0 — Usage of `LowCardinality` is restricted.

Default value: 1.

## allow\_suspicious\_low\_cardinality\_types

Allows or restricts using [LowCardinality](#) with data types with fixed size of 8 bytes or less: numeric data types and `FixedString(8_bytes_or_less)`.

For small fixed values using of `LowCardinality` is usually inefficient, because ClickHouse stores a numeric index for each row. As a result:

- Disk space usage can rise.
- RAM consumption can be higher, depending on a dictionary size.
- Some functions can work slower due to extra coding/encoding operations.

Merge times in `MergeTree`-engine tables can grow due to all the reasons described above.

Possible values:

- 1 — Usage of `LowCardinality` is not restricted.
- 0 — Usage of `LowCardinality` is restricted.

Default value: 0.

## `min_insert_block_size_rows_for_materialized_views`

Sets the minimum number of rows in the block which can be inserted into a table by an `INSERT` query. Smaller-sized blocks are squashed into bigger ones. This setting is applied only for blocks inserted into `materialized view`. By adjusting this setting, you control blocks squashing while pushing to materialized view and avoid excessive memory usage.

Possible values:

- Any positive integer.
- 0 — Squashing disabled.

Default value: 1048576.

### **See Also**

- [min\\_insert\\_block\\_size\\_rows](#)

## `min_insert_block_size_bytes_for_materialized_views`

Sets the minimum number of bytes in the block which can be inserted into a table by an `INSERT` query. Smaller-sized blocks are squashed into bigger ones. This setting is applied only for blocks inserted into `materialized view`. By adjusting this setting, you control blocks squashing while pushing to materialized view and avoid excessive memory usage.

Possible values:

- Any positive integer.
- 0 — Squashing disabled.

Default value: 268435456.

### **See also**

- [min\\_insert\\_block\\_size\\_bytes](#)

## `output_format_pretty_grid_charset`

Allows changing a charset which is used for printing grids borders. Available charsets are `UTF-8`, `ASCII`.

### **Example**

```
SET output_format.pretty_grid_charset = 'UTF-8';
SELECT * FROM a;
```

a
1

```
SET output_format.pretty_grid_charset = 'ASCII';
SELECT * FROM a;
+---+
| 1 |
+---+
```

## optimize\_read\_in\_order

Enables [ORDER BY](#) optimization in [SELECT](#) queries for reading data from [MergeTree](#) tables.

Possible values:

- 0 — [ORDER BY](#) optimization is disabled.
- 1 — [ORDER BY](#) optimization is enabled.

Default value: 1.

### See Also

- [ORDER BY Clause](#)

## optimize\_aggregation\_in\_order

Enables [GROUP BY](#) optimization in [SELECT](#) queries for aggregating data in corresponding order in [MergeTree](#) tables.

Possible values:

- 0 — [GROUP BY](#) optimization is disabled.
- 1 — [GROUP BY](#) optimization is enabled.

Default value: 0.

### See Also

- [GROUP BY optimization](#)

## mutations\_sync

Allows to execute [ALTER TABLE ... UPDATE|DELETE](#) queries ([mutations](#)) synchronously.

Possible values:

- 0 - Mutations execute asynchronously.
- 1 - The query waits for all mutations to complete on the current server.
- 2 - The query waits for all mutations to complete on all replicas (if they exist).

Default value: 0.

### See Also

- [Synchronicity of ALTER Queries](#)
- [Mutations](#)

## `ttl_only_drop_parts`

Enables or disables complete dropping of data parts where all rows are expired in [MergeTree](#) tables.

When `ttl_only_drop_parts` is disabled (by default), the ClickHouse server only deletes expired rows according to their TTL.

When `ttl_only_drop_parts` is enabled, the ClickHouse server drops a whole part when all rows in it are expired.

Dropping whole parts instead of partial cleaning TTL-d rows allows having shorter `merge_with_ttl_timeout` times and lower impact on system performance.

Possible values:

- 0 — The complete dropping of data parts is disabled.
- 1 — The complete dropping of data parts is enabled.

Default value: 0.

### **See Also**

- [CREATE TABLE query clauses and settings](#) (`merge_with_ttl_timeout` setting)
- [Table TTL](#)

## `lock_acquire_timeout`

Defines how many seconds a locking request waits before failing.

Locking timeout is used to protect from deadlocks while executing read/write operations with tables. When the timeout expires and the locking request fails, the ClickHouse server throws an exception "Locking attempt timed out! Possible deadlock avoided. Client should retry." with error code `DEADLOCK_AVOIDED`.

Possible values:

- Positive integer (in seconds).
- 0 — No locking timeout.

Default value: 120 seconds.

## `cast_keep_nullable`

Enables or disables keeping of the `Nullable` data type in [CAST](#) operations.

When the setting is enabled and the argument of `CAST` function is `Nullable`, the result is also transformed to `Nullable` type. When the setting is disabled, the result always has the destination type exactly.

Possible values:

- 0 — The `CAST` result has exactly the destination type specified.
- 1 — If the argument type is `Nullable`, the `CAST` result is transformed to `Nullable(DestinationDataType)`.

Default value: 0.

### **Examples**

The following query results in the destination data type exactly:

```
SET cast_keep_nullable = 0;
SELECT CAST(toNullable(toInt32(0)) AS Int32) as x, toTypeName(x);
```

Result:

```
x  toTypeName(CAST(toNullable(toInt32(0)), 'Int32'))  |
0 | Int32  |
```

The following query results in the `Nullable` modification on the destination data type:

```
SET cast_keep_nullable = 1;
SELECT CAST(toNullable(toInt32(0)) AS Int32) as x, toTypeName(x);
```

Result:

```
x  toTypeName(CAST(toNullable(toInt32(0)), 'Int32'))  |
0 | Nullable(Int32)  |
```

## See Also

- [CAST](#) function

## output\_format\_pretty\_max\_value\_width

Limits the width of value displayed in [Pretty](#) formats. If the value width exceeds the limit, the value is cut.

Possible values:

- Positive integer.
- 0 — The value is cut completely.

Default value: `10000` symbols.

## Examples

Query:

```
SET output_format_pretty_max_value_width = 10;
SELECT range(number) FROM system.numbers LIMIT 10 FORMAT PrettyCompactNoEscapes;
```

Result:

```
range(number)  |
[]  |
[0]  |
[0,1]  |
[0,1,2]  |
[0,1,2,3]  |
[0,1,2,3,4...  |
[0,1,2,3,4...  |
[0,1,2,3,4...  |
[0,1,2,3,4...  |
[0,1,2,3,4...]
```

Query with zero width:

```
SET output_format_pretty_max_value_width = 0;
SELECT range(number) FROM system.numbers LIMIT 5 FORMAT PrettyCompactNoEscapes;
```

Result:

range(number)
...
...
...
...
...

## output\_format\_pretty\_row\_numbers

Adds row numbers to output in the [Pretty](#) format.

Possible values:

- 0 — Output without row numbers.
- 1 — Output with row numbers.

Default value: 0.

### Example

Query:

```
SET output_format_pretty_row_numbers = 1;
SELECT TOP 3 name, value FROM system.settings;
```

Result:

	name	value
1.	min_compress_block_size	65536
2.	max_compress_block_size	1048576
3.	max_block_size	65505

## system\_events\_show\_zero\_values

Allows to select zero-valued events from `system.events`.

Some monitoring systems require passing all the metrics values to them for each checkpoint, even if the metric value is zero.

Possible values:

- 0 — Disabled.
- 1 — Enabled.

Default value: 0.

### Examples

Query

```
SELECT * FROM system.events WHERE event='QueryMemoryLimitExceeded';
```

## Result

```
Ok.
```

## Query

```
SET system_events_show_zero_values = 1;
SELECT * FROM system.events WHERE event='QueryMemoryLimitExceeded';
```

## Result

event	value	description
QueryMemoryLimitExceeded	0	Number of times when memory limit exceeded for query.

## persistent

Disables persistency for the [Set](#) and [Join](#) table engines.

Reduces the I/O overhead. Suitable for scenarios that pursue performance and do not require persistence.

Possible values:

- 1 — Enabled.
- 0 — Disabled.

Default value: 1.

## output\_format\_csv\_null\_representation

Defines the representation of NULL for [CSV](#) output format. User can set any string as a value, for example, [My NULL](#).

Default value: \N.

## Examples

### Query

```
SELECT * from csv_custom_null FORMAT CSV;
```

### Result

```
788
\N
\N
```

### Query

```
SET output_format_csv_null_representation = 'My NULL';
SELECT * FROM csv_custom_null FORMAT CSV;
```

## Result

```
788
My NULL
My NULL
```

## output\_format\_tsv\_null\_representation

Defines the representation of `NULL` for `TSV` output format. User can set any string as a value, for example, `My NULL`.

Default value: `\N`.

### Examples

#### Query

```
SELECT * FROM tsv_custom_null FORMAT TSV;
```

#### Result

```
788
\N
\N
```

#### Query

```
SET output_format_tsv_null_representation = 'My NULL';
SELECT * FROM tsv_custom_null FORMAT TSV;
```

#### Result

```
788
My NULL
My NULL
```

## output\_format\_json\_array\_of\_rows

Enables the ability to output all rows as a JSON array in the `JSONEachRow` format.

Possible values:

- 1 — ClickHouse outputs all rows as an array, each row in the `JSONEachRow` format.
- 0 — ClickHouse outputs each row separately in the `JSONEachRow` format.

Default value: 0.

### Example of a query with the enabled setting

Query:

```
SET output_format_json_array_of_rows = 1;
SELECT number FROM numbers(3) FORMAT JSONEachRow;
```

Result:

```
[  
 {"number":"0"},  
 {"number":"1"},  
 {"number":"2"}  
]
```

## Example of a query with the disabled setting

Query:

```
SET output_format_json_array_of_rows = 0;  
SELECT number FROM numbers(3) FORMAT JSONEachRow;
```

Result:

```
{"number":"0"}  
 {"number":"1"}  
 {"number":"2"}
```

## allow\_nullable\_key

Allows using of the **Nullable**-typed values in a sorting and a primary key for **MergeTree** tables.

Possible values:

- 1 — **Nullable**-type expressions are allowed in keys.
- 0 — **Nullable**-type expressions are not allowed in keys.

Default value: 0.

## aggregate\_functions\_null\_for\_empty

Enables or disables rewriting all aggregate functions in a query, adding **-OrNull** suffix to them. Enable it for SQL standard compatibility.

It is implemented via query rewrite (similar to [count\\_distinctImplementation](#) setting) to get consistent results for distributed queries.

Possible values:

- 0 — Disabled.
- 1 — Enabled.

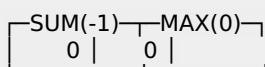
Default value: 0.

## Example

Consider the following query with aggregate functions:

```
SELECT SUM(-1), MAX(0) FROM system.one WHERE 0;
```

With **aggregate\_functions\_null\_for\_empty = 0** it would produce:



With `aggregate_functions_null_for_empty = 1` the result would be:

SUMOrNull(-1)	MAXOrNull(0)
NULL	NULL

## union\_default\_mode

Sets a mode for combining `SELECT` query results. The setting is only used when shared with `UNION` without explicitly specifying the `UNION ALL` or `UNION DISTINCT`.

Possible values:

- `'DISTINCT'` — ClickHouse outputs rows as a result of combining queries removing duplicate rows.
- `'ALL'` — ClickHouse outputs all rows as a result of combining queries including duplicate rows.
- `""` — ClickHouse generates an exception when used with `UNION`.

Default value: `"`.

See examples in [UNION](#).

## data\_type\_default\_nullable

Allows data types without explicit modifiers `NONE` or `NOT NULL` in column definition will be `Nullable`.

Possible values:

- 1 — The data types in column definitions are set to `Nullable` by default.
- 0 — The data types in column definitions are set to not `Nullable` by default.

Default value: 0.

## execute\_merges\_on\_single\_replica\_time\_threshold

Enables special logic to perform merges on replicas.

Possible values:

- Positive integer (in seconds).
- 0 — Special merges logic is not used. Merges happen in the usual way on all the replicas.

Default value: 0.

## Usage

Selects one replica to perform the merge on. Sets the time threshold from the start of the merge. Other replicas wait for the merge to finish, then download the result. If the time threshold passes and the selected replica does not perform the merge, then the merge is performed on other replicas as usual.

High values for that threshold may lead to replication delays.

It can be useful when merges are CPU bounded not IO bounded (performing heavy data compression, calculating aggregate functions or default expressions that require a large amount of calculations, or just very high number of tiny merges).

## max\_final\_threads

Sets the maximum number of parallel threads for the `SELECT` query data read phase with the **FINAL** modifier.

Possible values:

- Positive integer.
- 0 or 1 — Disabled. `SELECT` queries are executed in a single thread.

Default value: 16.

## opentelemetry\_start\_trace\_probability

Sets the probability that the ClickHouse can start a trace for executed queries (if no parent **trace context** is supplied).

Possible values:

- 0 — The trace for all executed queries is disabled (if no parent trace context is supplied).
- Positive floating-point number in the range [0..1]. For example, if the setting value is 0,5, ClickHouse can start a trace on average for half of the queries.
- 1 — The trace for all executed queries is enabled.

Default value: 0.

## optimize\_on\_insert

Enables or disables data transformation before the insertion, as if `merge` was done on this block (according to table engine).

Possible values:

- 0 — Disabled.
- 1 — Enabled.

Default value: 1.

### Example

The difference between enabled and disabled:

Query:

```
SET optimize_on_insert = 1;

CREATE TABLE test1 (`FirstTable` UInt32) ENGINE = ReplacingMergeTree ORDER BY FirstTable;
INSERT INTO test1 SELECT number % 2 FROM numbers(5);
SELECT * FROM test1;
SET optimize_on_insert = 0;

CREATE TABLE test2 (`SecondTable` UInt32) ENGINE = ReplacingMergeTree ORDER BY SecondTable;
INSERT INTO test2 SELECT number % 2 FROM numbers(5);
SELECT * FROM test2;
```

Result:

FirstTable	
0	
1	

SecondTable	
0	
0	
0	
1	
1	

Note that this setting influences [Materialized view](#) and [MaterializedMySQL](#) behaviour.

## engine\_file\_empty\_if\_not\_exists

Allows to select data from a file engine table without file.

Possible values:

- 0 — `SELECT` throws exception.
- 1 — `SELECT` returns empty result.

Default value: 0.

## engine\_file\_truncate\_on\_insert

Enables or disables truncate before insert in [File](#) engine tables.

Possible values:

- 0 — `INSERT` query appends new data to the end of the file.
- 1 — `INSERT` replaces existing content of the file with the new data.

Default value: 0.

## allow\_experimental\_geo\_types

Allows working with experimental [geo data types](#).

Possible values:

- 0 — Working with geo data types is disabled.
- 1 — Working with geo data types is enabled.

Default value: 0.

## database\_atomic\_wait\_for\_drop\_and\_detach\_synchronously

Adds a modifier `SYNC` to all `DROP` and `DETACH` queries.

Possible values:

- 0 — Queries will be executed with delay.
- 1 — Queries will be executed without delay.

Default value: 0.

## show\_table\_uuid\_in\_table\_create\_query\_if\_not\_nil

Sets the `SHOW TABLE` query display.

Possible values:

- 0 — The query will be displayed without table UUID.
- 1 — The query will be displayed with table UUID.

Default value: 0.

## allow\_experimental\_live\_view

Allows creation of experimental [live views](#).

Possible values:

- 0 — Working with live views is disabled.
- 1 — Working with live views is enabled.

Default value: 0.

## live\_view\_heartbeat\_interval

Sets the heartbeat interval in seconds to indicate [live view](#) is alive .

Default value: 15.

## max\_live\_view\_insert\_blocks\_before\_refresh

Sets the maximum number of inserted blocks after which mergeable blocks are dropped and query for [live view](#) is re-executed.

Default value: 64.

## temporary\_live\_view\_timeout

Sets the interval in seconds after which [live view](#) with timeout is deleted.

Default value: 5.

## periodic\_live\_view\_refresh

Sets the interval in seconds after which periodically refreshed [live view](#) is forced to refresh.

Default value: 60.

## http\_connection\_timeout

HTTP connection timeout (in seconds).

Possible values:

- Any positive integer.
- 0 - Disabled (infinite timeout).

Default value: 1.

## http\_send\_timeout

HTTP send timeout (in seconds).

Possible values:

- Any positive integer.
- 0 - Disabled (infinite timeout).

Default value: 1800.

## http\_receive\_timeout

HTTP receive timeout (in seconds).

Possible values:

- Any positive integer.
- 0 - Disabled (infinite timeout).

Default value: 1800.

## check\_query\_single\_value\_result

Defines the level of detail for the [CHECK TABLE](#) query result for MergeTree family engines .

Possible values:

- 0 — the query shows a check status for every individual data part of a table.
- 1 — the query shows the general table check status.

Default value: 0.

## prefer\_column\_name\_to\_alias

Enables or disables using the original column names instead of aliases in query expressions and clauses. It especially matters when alias is the same as the column name, see [Expression Aliases](#). Enable this setting to make aliases syntax rules in ClickHouse more compatible with most other database engines.

Possible values:

- 0 — The column name is substituted with the alias.
- 1 — The column name is not substituted with the alias.

Default value: 0.

### Example

The difference between enabled and disabled:

Query:

```
SET prefer_column_name_to_alias = 0;
SELECT avg(number) AS number, max(number) FROM numbers(10);
```

Result:

```
Received exception from server (version 21.5.1):
Code: 184. DB::Exception: Received from localhost:9000. DB::Exception: Aggregate function avg(number) is found
inside another aggregate function in query: While processing avg(number) AS number.
```

Query:

```
SET prefer_column_name_to_alias = 1;
SELECT avg(number) AS number, max(number) FROM numbers(10);
```

Result:

number	max(number)
4.5	9

## limit

Sets the maximum number of rows to get from the query result. It adjusts the value set by the **LIMIT** clause, so that the limit, specified in the query, cannot exceed the limit, set by this setting.

Possible values:

- 0 — The number of rows is not limited.
- Positive integer.

Default value: 0.

## offset

Sets the number of rows to skip before starting to return rows from the query. It adjusts the offset set by the **OFFSET** clause, so that these two values are summarized.

Possible values:

- 0 — No rows are skipped .
- Positive integer.

Default value: 0.

## Example

Input table:

```
CREATE TABLE test (i UInt64) ENGINE = MergeTree() ORDER BY i;
INSERT INTO test SELECT number FROM numbers(500);
```

Query:

```
SET limit = 5;
SET offset = 7;
SELECT * FROM test LIMIT 10 OFFSET 100;
```

Result:

i
107
108
109

## optimize\_syntax\_fuse\_functions

Enables to fuse aggregate functions with identical argument. It rewrites query contains at least two aggregate functions from `sum`, `count` or `avg` with identical argument to `sumCount`.

Possible values:

- 0 — Functions with identical argument are not fused.
- 1 — Functions with identical argument are fused.

Default value: 0.

### Example

Query:

```
CREATE TABLE fuse_tbl(a Int8, b Int8) Engine = Log;
SET optimize_syntax_fuse_functions = 1;
EXPLAIN SYNTAX SELECT sum(a), sum(b), count(b), avg(b) from fuse_tbl FORMAT TSV;
```

Result:

```
SELECT
    sum(a),
    sumCount(b).1,
    sumCount(b).2,
    (sumCount(b).1) / (sumCount(b).2)
FROM fuse_tbl
```

## allow\_experimental\_database\_replicated

Enables to create databases with `Replicated` engine.

Possible values:

- 0 — Disabled.
- 1 — Enabled.

Default value: 0.

## database\_replicated\_initial\_query\_timeout\_sec

Sets how long initial DDL query should wait for Replicated database to process previous DDL queue entries in seconds.

Possible values:

- Positive integer.
- 0 — Unlimited.

Default value: 300.

## distributed\_ddl\_task\_timeout

Sets timeout for DDL query responses from all hosts in cluster. If a DDL request has not been performed on all hosts, a response will contain a timeout error and a request will be executed in an async mode.

Negative value means infinite.

Possible values:

- Positive integer.
- 0 — Async mode.
- Negative integer — infinite timeout.

Default value: 180.

## distributed\_ddl\_output\_mode

Sets format of distributed DDL query result.

Possible values:

- `throw` — Returns result set with query execution status for all hosts where query is finished. If query has failed on some hosts, then it will rethrow the first exception. If query is not finished yet on some hosts and `distributed_ddl_task_timeout` exceeded, then it throws `TIMEOUT_EXCEEDED` exception.
- `none` — Is similar to `throw`, but distributed DDL query returns no result set.
- `null_status_on_timeout` — Returns `NULL` as execution status in some rows of result set instead of throwing `TIMEOUT_EXCEEDED` if query is not finished on the corresponding hosts.
- `never_throw` — Do not throw `TIMEOUT_EXCEEDED` and do not rethrow exceptions if query has failed on some hosts.

Default value: `throw`.

## flatten\_nested

Sets the data format of a `nested` columns.

Possible values:

- 1 — Nested column is flattened to separate arrays.
- 0 — Nested column stays a single array of tuples.

Default value: 1.

### Usage

If the setting is set to 0, it is possible to use an arbitrary level of nesting.

### Examples

Query:

```
SET flatten_nested = 1;
CREATE TABLE t_nest (`n` Nested(a UInt32, b UInt32)) ENGINE = MergeTree ORDER BY tuple();

SHOW CREATE TABLE t_nest;
```

Result:

#### statement

```
| CREATE TABLE default.t_nest
(
    `n.a` Array(UInt32),
    `n.b` Array(UInt32)
)
ENGINE = MergeTree
ORDER BY tuple()
SETTINGS index_granularity = 8192 |
```

Query:

```
SET flatten_nested = 0;

CREATE TABLE t_nest (`n` Nested(a UInt32, b UInt32)) ENGINE = MergeTree ORDER BY tuple();

SHOW CREATE TABLE t_nest;
```

Result:

#### statement

```
| CREATE TABLE default.t_nest
(
    `n` Nested(a UInt32, b UInt32)
)
ENGINE = MergeTree
ORDER BY tuple()
SETTINGS index_granularity = 8192 |
```

## external\_table\_functions\_use\_nulls

Defines how [mysql](#), [postgresql](#) and [odbc](#)] table functions use Nullable columns.

Possible values:

- 0 — The table function explicitly uses Nullable columns.
- 1 — The table function implicitly uses Nullable columns.

Default value: 1.

#### Usage

If the setting is set to 0, the table function does not make Nullable columns and inserts default values instead of NULL. This is also applicable for NULL values inside arrays.

## output\_format\_arrow\_low\_cardinality\_as\_dictionary

Allows to convert the [LowCardinality](#) type to the [DICTIONARY](#) type of the [Arrow](#) format for [SELECT](#) queries.

Possible values:

- 0 — The [LowCardinality](#) type is not converted to the [DICTIONARY](#) type.
- 1 — The [LowCardinality](#) type is converted to the [DICTIONARY](#) type.

Default value: 0.

## materialized\_postgresql\_max\_block\_size

Sets the number of rows collected in memory before flushing data into PostgreSQL database table.

Possible values:

- Positive integer.

Default value: 65536.

## materialized\_postgresql\_tables\_list

Sets a comma-separated list of PostgreSQL database tables, which will be replicated via [MaterializedPostgreSQL](#) database engine.

Default value: empty list — means whole PostgreSQL database will be replicated.

## materialized\_postgresql\_allow\_automatic\_update

Allows reloading table in the background, when schema changes are detected. DDL queries on the PostgreSQL side are not replicated via ClickHouse [MaterializedPostgreSQL](#) engine, because it is not allowed with PostgreSQL logical replication protocol, but the fact of DDL changes is detected transactionally. In this case, the default behaviour is to stop replicating those tables once DDL is detected. However, if this setting is enabled, then, instead of stopping the replication of those tables, they will be reloaded in the background via database snapshot without data losses and replication will continue for them.

Possible values:

- 0 — The table is not automatically updated in the background, when schema changes are detected.
- 1 — The table is automatically updated in the background, when schema changes are detected.

Default value: 0.

## materialized\_postgresql\_replication\_slot

A user-created replication slot. Must be used together with [materialized\\_postgresql\\_snapshot](#).

## materialized\_postgresql\_snapshot

A text string identifying a snapshot, from which [initial dump of PostgreSQL tables](#) will be performed. Must be used together with [materialized\\_postgresql\\_replication\\_slot](#).

## allow\_experimental\_projection\_optimization

Enables or disables [projection](#) optimization when processing SELECT queries.

Possible values:

- 0 — Projection optimization disabled.
- 1 — Projection optimization enabled.

Default value: 0.

## force\_optimize\_projection

Enables or disables the obligatory use of [projections](#) in [SELECT](#) queries, when projection optimization is enabled (see [allow\\_experimental\\_projection\\_optimization](#) setting).

Possible values:

- 0 — Projection optimization is not obligatory.
- 1 — Projection optimization is obligatory.

Default value: 0.

## replication\_alter\_partitions\_sync

Allows to set up waiting for actions to be executed on replicas by [ALTER](#), [OPTIMIZE](#) or [TRUNCATE](#) queries.

Possible values:

- 0 — Do not wait.
- 1 — Wait for own execution.
- 2 — Wait for everyone.

Default value: 1.

## replication\_wait\_for\_inactive\_replica\_timeout

Specifies how long (in seconds) to wait for inactive replicas to execute [ALTER](#), [OPTIMIZE](#) or [TRUNCATE](#) queries.

Possible values:

- 0 — Do not wait.
- Negative integer — Wait for unlimited time.
- Positive integer — The number of seconds to wait.

Default value: 120 seconds.

## regexp\_max\_matches\_per\_row

Sets the maximum number of matches for a single regular expression per row. Use it to protect against memory overload when using greedy regular expression in the [extractAllGroupsHorizontal](#) function.

Possible values:

- Positive integer.

Default value: 1000.

## http\_max\_single\_read\_retries

Sets the maximum number of retries during a single HTTP read.

Possible values:

- Positive integer.

Default value: 1024.

## log\_queries\_probability

Allows a user to write to `query_log`, `query_thread_log`, and `query_views_log` system tables only a sample of queries selected randomly with the specified probability. It helps to reduce the load with a large volume of queries in a second.

Possible values:

- 0 — Queries are not logged in the system tables.
- Positive floating-point number in the range [0..1]. For example, if the setting value is 0.5, about half of the queries are logged in the system tables.
- 1 — All queries are logged in the system tables.

Default value: 1.

## short\_circuit\_function\_evaluation

Allows calculating the `if`, `multilf`, `and`, and `or` functions according to a [short scheme](#). This helps optimize the execution of complex expressions in these functions and prevent possible exceptions (such as division by zero when it is not expected).

Possible values:

- `enable` — Enables short-circuit function evaluation for functions that are suitable for it (can throw an exception or computationally heavy).
- `force_enable` — Enables short-circuit function evaluation for all functions.
- `disable` — Disables short-circuit function evaluation.

Default value: `enable`.

## max\_hyperscan\_regexp\_length

Defines the maximum length for each regular expression in the [hyperscan multi-match functions](#).

Possible values:

- Positive integer.
- 0 - The length is not limited.

Default value: 0.

### Example

Query:

```
SELECT multiMatchAny('abcd', ['ab','bcd','c','d']) SETTINGS max_hyperscan_regexp_length = 3;
```

Result:

```
multiMatchAny('abcd', ['ab', 'bcd', 'c', 'd'])  
1 |
```

Query:

```
SELECT multiMatchAny('abcd', ['ab','bcd','c','d']) SETTINGS max_hyperscan_regexp_length = 2;
```

Result:

```
Exception: Regexp length too large.
```

## See Also

- [max\\_hyperscan\\_regex\\_total\\_length](#)

## max\_hyperscan\_regex\_total\_length

Sets the maximum length total of all regular expressions in each [hyperscan multi-match function](#).

Possible values:

- Positive integer.
- 0 - The length is not limited.

Default value: 0.

## Example

Query:

```
SELECT multiMatchAny('abcd', ['a','b','c','d']) SETTINGS max_hyperscan_regex_total_length = 5;
```

Result:

```
multiMatchAny('abcd', ['a', 'b', 'c', 'd'])  
1 |
```

Query:

```
SELECT multiMatchAny('abcd', ['ab','bc','c','d']) SETTINGS max_hyperscan_regex_total_length = 5;
```

Result:

```
Exception: Total regexp lengths too large.
```

## See Also

- [max\\_hyperscan\\_regex\\_length](#)

## enable\_positional\_arguments

Enables or disables supporting positional arguments for [GROUP BY](#), [LIMIT BY](#), [ORDER BY](#) statements. When you want to use column numbers instead of column names in these clauses, set `enable_positional_arguments = 1`.

Possible values:

- 0 — Positional arguments aren't supported.
- 1 — Positional arguments are supported: column numbers can use instead of column names.

Default value: 0.

## Example

Query:

```
CREATE TABLE positional_arguments(one Int, two Int, three Int) ENGINE=Memory();
INSERT INTO positional_arguments VALUES (10, 20, 30), (20, 20, 10), (30, 10, 20);
SET enable_positional_arguments = 1;
SELECT * FROM positional_arguments ORDER BY 2,3;
```

Result:

one	two	three
30	10	20
20	20	10
10	20	30

## optimize\_move\_to\_prewhere

Enables or disables automatic **PREWHERE** optimization in **SELECT** queries.

Works only for **\*MergeTree** tables.

Possible values:

- 0 — Automatic **PREWHERE** optimization is disabled.
- 1 — Automatic **PREWHERE** optimization is enabled.

Default value: 1.

## optimize\_move\_to\_prewhere\_if\_final

Enables or disables automatic **PREWHERE** optimization in **SELECT** queries with **FINAL** modifier.

Works only for **\*MergeTree** tables.

Possible values:

- 0 — Automatic **PREWHERE** optimization in **SELECT** queries with **FINAL** modifier is disabled.
- 1 — Automatic **PREWHERE** optimization in **SELECT** queries with **FINAL** modifier is enabled.

Default value: 0.

## See Also

- [optimize\\_move\\_to\\_prewhere](#) setting

## describe\_include\_subcolumns

Enables describing subcolumns for a **DESCRIBE** query. For example, members of a **Tuple** or subcolumns of a **Map**, **Nullable** or an **Array** data type.

Possible values:

- 0 — Subcolumns are not included in **DESCRIBE** queries.
- 1 — Subcolumns are included in **DESCRIBE** queries.

Default value: 0.

## Example

See an example for the `DESCRIBE` statement.

## async\_insert

Enables or disables asynchronous inserts. This makes sense only for insertion over HTTP protocol. Note that deduplication isn't working for such inserts.

If enabled, the data is combined into batches before the insertion into tables, so it is possible to do small and frequent insertions into ClickHouse (up to 15000 queries per second) without buffer tables.

The data is inserted either after the `async_insert_max_data_size` is exceeded or after `async_insert_busy_timeout_ms` milliseconds since the first `INSERT` query. If the `async_insert_stale_timeout_ms` is set to a non-zero value, the data is inserted after `async_insert_stale_timeout_ms` milliseconds since the last query.

If `wait_for_async_insert` is enabled, every client will wait for the data to be processed and flushed to the table. Otherwise, the query would be processed almost instantly, even if the data is not inserted.

Possible values:

- 0 — Insertions are made synchronously, one after another.
- 1 — Multiple asynchronous insertions enabled.

Default value: 0.

## async\_insert\_threads

The maximum number of threads for background data parsing and insertion.

Possible values:

- Positive integer.
- 0 — Asynchronous insertions are disabled.

Default value: 16.

## wait\_for\_async\_insert

Enables or disables waiting for processing of asynchronous insertion. If enabled, server will return `OK` only after the data is inserted. Otherwise, it will return `OK` even if the data wasn't inserted.

Possible values:

- 0 — Server returns `OK` even if the data is not yet inserted.
- 1 — Server returns `OK` only after the data is inserted.

Default value: 1.

## wait\_for\_async\_insert\_timeout

The timeout in seconds for waiting for processing of asynchronous insertion.

Possible values:

- Positive integer.

- 0 — Disabled.

Default value: `lock_acquire_timeout`.

## async\_insert\_max\_data\_size

The maximum size of the unparsed data in bytes collected per query before being inserted.

Possible values:

- Positive integer.
- 0 — Asynchronous insertions are disabled.

Default value: `1000000`.

## async\_insert\_busy\_timeout\_ms

The maximum timeout in milliseconds since the first `INSERT` query before inserting collected data.

Possible values:

- Positive integer.
- 0 — Timeout disabled.

Default value: `200`.

## async\_insert\_stale\_timeout\_ms

The maximum timeout in milliseconds since the last `INSERT` query before dumping collected data. If enabled, the settings prolongs the `async_insert_busy_timeout_ms` with every `INSERT` query as long as `async_insert_max_data_size` is not exceeded.

Possible values:

- Positive integer.
- 0 — Timeout disabled.

Default value: `0`.

---

# ClickHouse Utility

- `clickhouse-local` — Allows running SQL queries on data without stopping the ClickHouse server, similar to how `awk` does this.
- `clickhouse-copier` — Copies (and reshards) data from one cluster to another cluster.
- `clickhouse-benchmark` — Loads server with the custom queries and settings.
- `clickhouse-format` — Enables formatting input queries.
- `ClickHouse obfuscator` — Obfuscates data.
- `ClickHouse compressor` — Compresses and decompresses data.
- `clickhouse-odbc-bridge` — A proxy server for ODBC driver.

---

## clickhouse-copier

Copies data from the tables in one cluster to tables in another (or the same) cluster.

## Warning

To get a consistent copy, the data in the source tables and partitions should not change during the entire process.

You can run multiple `clickhouse-copier` instances on different servers to perform the same job. ZooKeeper is used for syncing the processes.

After starting, `clickhouse-copier`:

- Connects to ZooKeeper and receives:
  - Copying jobs.
  - The state of the copying jobs.
- It performs the jobs.

Each running process chooses the “closest” shard of the source cluster and copies the data into the destination cluster, resharding the data if necessary.

`clickhouse-copier` tracks the changes in ZooKeeper and applies them on the fly.

To reduce network traffic, we recommend running `clickhouse-copier` on the same server where the source data is located.

## Running Clickhouse-copier

The utility should be run manually:

```
$ clickhouse-copier --daemon --config zookeeper.xml --task-path /task/path --base-dir /path/to/dir
```

Parameters:

- `daemon` — Starts `clickhouse-copier` in daemon mode.
- `config` — The path to the `zookeeper.xml` file with the parameters for the connection to ZooKeeper.
- `task-path` — The path to the ZooKeeper node. This node is used for syncing `clickhouse-copier` processes and storing tasks. Tasks are stored in `$task-path/description`.
- `task-file` — Optional path to file with task configuration for initial upload to ZooKeeper.
- `task-upload-force` — Force upload `task-file` even if node already exists.
- `base-dir` — The path to logs and auxiliary files. When it starts, `clickhouse-copier` creates `clickhouse-copier_YYYYMMHHSS_<PID>` subdirectories in `$base-dir`. If this parameter is omitted, the directories are created in the directory where `clickhouse-copier` was launched.

## Format of Zookeeper.xml

```

<clickhouse>
  <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>
</clickhouse>

```

## Configuration of Copying Tasks

```

<clickhouse>
  <!-- Configuration of clusters as in an ordinary server config -->
  <remote_servers>
    <source_cluster>
      <!--
        source cluster & destination clusters accept exactly the same
        parameters as parameters for the usual Distributed table
        see https://clickhouse.com/docs/en/engines/table-engines/special/distributed/
      -->
      <shard>
        <internal_replication>false</internal_replication>
        <replica>
          <host>127.0.0.1</host>
          <port>9000</port>
          <!--
            <user>default</user>
            <password>default</password>
            <secure>1</secure>
          -->
        </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.

        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>
</clickhouse>

```

clickhouse-copier tracks the changes in /task/path/description and applies them on the fly. For instance, if you change the value of max\_workers, the number of processes running tasks will also change.

## clickhouse-local

The clickhouse-local program enables you to perform fast processing on local files, without having to deploy and configure the ClickHouse server.

Accepts data that represent tables and queries them using [ClickHouse SQL dialect](#).

`clickhouse-local` uses the same core as ClickHouse server, so it supports most of the features and the same set of formats and table engines.

By default `clickhouse-local` does not have access to data on the same host, but it supports loading server configuration using `--config-file` argument.

## Warning

It is not recommended to load production server configuration into `clickhouse-local` because data can be damaged in case of human error.

For temporary data, a unique temporary data directory is created by default.

## Usage

Basic usage:

```
$ clickhouse-local --structure "table_structure" --input-format "format_of_incoming_data" \
--query "query"
```

Arguments:

- `-S, --structure` — table structure for input data.
- `-if, --input-format` — input format, `TSV` by default.
- `-f, --file` — path to data, `stdin` by default.
- `-q, --query` — queries to execute with ; as delimiter. You must specify either `query` or `queries-file` option.
- `-qf, --queries-file` - file path with queries to execute. You must specify either `query` or `queries-file` option.
- `-N, --table` — table name where to put output data, `table` by default.
- `-of, --format, --output-format` — output format, `TSV` by default.
- `-d, --database` — default database, `_local` by default.
- `--stacktrace` — whether to dump debug output in case of exception.
- `--echo` — print query before execution.
- `--verbose` — more details on query execution.
- `--logger.console` — Log to console.
- `--logger.log` — Log file name.
- `--logger.level` — Log level.
- `--ignore-error` — do not stop processing if a query failed.
- `-c, --config-file` — path to configuration file in same format as for ClickHouse server, by default the configuration empty.
- `--no-system-tables` — do not attach system tables.
- `--help` — arguments references for `clickhouse-local`.

- `-V, --version` — print version information and exit.

Also there are arguments for each ClickHouse configuration variable which are more commonly used instead of `--config-file`.

## Examples

```
$ echo -e "1,2\n3,4" | clickhouse-local --structure "a Int64, b Int64" \
--input-format "CSV" --query "SELECT * FROM table"
Read 2 rows, 32.00 B in 0.000 sec., 5182 rows/sec., 80.97 KiB/sec.
1 2
3 4
```

Previous example is the same as:

```
$ echo -e "1,2\n3,4" | clickhouse-local --query "
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
```

You don't have to use `stdin` or `--file` argument, and can open any number of files using the file [table function](#):

```
$ echo 1 | tee 1.tsv
1

$ echo 2 | tee 2.tsv
2

$ clickhouse-local --query "
select * from file('1.tsv', TSV, 'a int') t1
cross join file('2.tsv', TSV, 'b int') t2"
1 2
```

Now let's output memory user for each Unix user:

Query:

```
$ ps aux | tail -n +2 | awk '{ printf("%s\t%s\n", $1, $4 ) }' \
| clickhouse-local --structure "user String, mem Float64" \
--query "SELECT user, round(sum(mem), 2) as memTotal
FROM table GROUP BY user ORDER BY memTotal DESC FORMAT Pretty"
```

Result:

```
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 |
+-----+-----+
...
```

## clickhouse-benchmark

Connects to a ClickHouse server and repeatedly sends specified queries.

## Syntax

```
$ clickhouse-benchmark --query ["single query"] [keys]
```

or

```
$ echo "single query" | clickhouse-benchmark [keys]
```

or

```
$ clickhouse-benchmark [keys] <<< "single query"
```

If you want to send a set of queries, create a text file and place each query on the individual string in this file. For example:

```
SELECT * FROM system.numbers LIMIT 10000000;  
SELECT 1;
```

Then pass this file to a standard input of `clickhouse-benchmark`:

```
clickhouse-benchmark [keys] < queries_file;
```

## Keys

- `--query=QUERY` — Query to execute. If this parameter is not passed, `clickhouse-benchmark` will read queries from standard input.
- `-c N, --concurrency=N` — Number of queries that `clickhouse-benchmark` sends simultaneously. Default value: 1.
- `-d N, --delay=N` — Interval in seconds between intermediate reports (to disable reports set 0). Default value: 1.
- `-h HOST, --host=HOST` — Server host. Default value: `localhost`. For the [comparison mode](#) you can use multiple `-h` keys.
- `-p N, --port=N` — Server port. Default value: 9000. For the [comparison mode](#) you can use multiple `-p` keys.
- `-i N, --iterations=N` — Total number of queries. Default value: 0 (repeat forever).
- `-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` stops sending queries when the specified time limit is reached. Default value: 0 (time limit disabled).
- `--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 [comparison mode](#) `clickhouse-benchmark` performs the [Independent two-sample Student's t-test](#) to determine whether the two distributions aren't different with the selected level of confidence.
- `--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` outputs a report to the specified JSON-file.
- `--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` outputs stack traces of exceptions.
- `--stage=WORD` — Query processing stage at server. ClickHouse stops query processing and returns an answer to `clickhouse-benchmark` at the specified stage. Possible values: `complete`, `fetch_columns`, `with_mergeable_state`. Default value: `complete`.
- `--help` — Shows the help message.

If you want to apply some **settings** for queries, pass them as a key `--<session setting name>=SETTING_VALUE`. For example, `--max_memory_usage=1048576`.

## Output

By default, `clickhouse-benchmark` reports for each `--delay` interval.

Example of the report:

```
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.
```

In the report you can find:

- Number of queries in the `Queries executed`: field.

- Status string containing (in order):
  - Endpoint of ClickHouse server.
  - Number of processed queries.
  - QPS: How many queries the server performed per second during a period specified in the `--delay` argument.
  - RPS: How many rows the server reads per second during a period specified in the `--delay` argument.
  - MiB/s: How many mebibytes the server reads per second during a period specified in the `--delay` argument.
  - result RPS: How many rows placed by the server to the result of a query per second during a period specified in the `--delay` argument.
  - result MiB/s. How many mebibytes placed by the server to the result of a query per second during a period specified in the `--delay` argument.
- Percentiles of queries execution time.

## Comparison Mode

`clickhouse-benchmark` can compare performances for two running ClickHouse servers.

To use the comparison mode, specify endpoints of both servers by two pairs of `--host`, `--port` keys. Keys matched together by position in arguments list, the first `--host` is matched with the first `--port` and so on. `clickhouse-benchmark` establishes connections to both servers, then sends queries. Each query addressed to a randomly selected server. The results are shown for each server separately.

## Example

```
$ 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-format

Allows formatting input queries.

Keys:

- `--help` or `-h` — Produce help message.
- `--hilite` — Add syntax highlight with ANSI terminal escape sequences.
- `--oneline` — Format in single line.
- `--quiet` or `-q` — Just check syntax, no output on success.
- `--multiquery` or `-n` — Allow multiple queries in the same file.
- `--obfuscate` — Obfuscate instead of formatting.
- `--seed <string>` — Seed arbitrary string that determines the result of obfuscation.
- `--backslash` — Add a backslash at the end of each line of the formatted query. Can be useful when you copy a query from web or somewhere else with multiple lines, and want to execute it in command line.

# Examples

1. Highlighting and single line:

```
$ clickhouse-format --oneline --hilite <<< "SELECT sum(number) FROM numbers(5);"
```

Result:

```
SELECT sum(number) FROM numbers(5)
```

2. Multiqueries:

```
$ clickhouse-format -n <<< "SELECT * FROM (SELECT 1 AS x UNION ALL SELECT 1 UNION DISTINCT SELECT 3);"
```

Result:

```
SELECT *
FROM
(
    SELECT 1 AS x
    UNION ALL
    SELECT 1
    UNION DISTINCT
    SELECT 3
)
;
```

3. Obfuscating:

```
$ clickhouse-format --seed Hello --obfuscate <<< "SELECT cost_first_screen BETWEEN a AND b, CASE WHEN x >= 123 THEN y ELSE NULL END;"
```

Result:

```
SELECT treasury_mammoth_hazelnut BETWEEN nutmeg AND span, CASE WHEN chive >= 116 THEN switching ELSE ANYTHING END;
```

Same query and another seed string:

```
$ clickhouse-format --seed World --obfuscate <<< "SELECT cost_first_screen BETWEEN a AND b, CASE WHEN x >= 123 THEN y ELSE NULL END;"
```

Result:

```
SELECT horse_tape_summer BETWEEN folklore AND moccasins, CASE WHEN intestine >= 116 THEN nonconformist ELSE FORESTRY END;
```

4. Adding backslash:

```
$ clickhouse-format --backslash <<< "SELECT * FROM (SELECT 1 AS x UNION ALL SELECT 1 UNION DISTINCT SELECT 3);"
```

Result:

```
SELECT * \
FROM \
(\ \
  SELECT 1 AS x \
UNION ALL \
  SELECT 1 \
UNION DISTINCT \
  SELECT 3 \
)
```

## ClickHouse compressor

Simple program for data compression and decompression.

### Examples

Compress data with LZ4:

```
$ ./clickhouse-compressor < input_file > output_file
```

Decompress data from LZ4 format:

```
$ ./clickhouse-compressor --decompress < input_file > output_file
```

Compress data with ZSTD at level 5:

```
$ ./clickhouse-compressor --codec 'ZSTD(5)' < input_file > output_file
```

Compress data with Delta of four bytes and ZSTD level 10.

```
$ ./clickhouse-compressor --codec 'Delta(4)' --codec 'ZSTD(10)' < input_file > output_file
```

## ClickHouse obfuscator

A simple tool for table data obfuscation.

It reads an input table and produces an output table, that retains some properties of input, but contains different data.

It allows publishing almost real production data for usage in benchmarks.

It is designed to retain the following properties of data:

- cardinalities of values (number of distinct values) for every column and every tuple of columns;
- conditional cardinalities: number of distinct values of one column under the condition on the value of another column;
- probability distributions of the absolute value of integers; the sign of signed integers; exponent and sign for floats;
- probability distributions of the length of strings;
- probability of zero values of numbers; empty strings and arrays, `NULLs`;
- data compression ratio when compressed with LZ77 and entropy family of codecs;
- continuity (magnitude of difference) of time values across the table; continuity of floating-point values;
- date component of `DateTime` values;

- UTF-8 validity of string values;
- string values look natural.

Most of the properties above are viable for performance testing:

reading data, filtering, aggregation, and sorting will work at almost the same speed as on original data due to saved cardinalities, magnitudes, compression ratios, etc.

It works in a deterministic fashion: you define a seed value and the transformation is determined by input data and by seed.

Some transformations are one to one and could be reversed, so you need to have a large seed and keep it in secret.

It uses some cryptographic primitives to transform data but from the cryptographic point of view, it does not do it properly, that is why you should not consider the result as secure unless you have another reason. The result may retain some data you don't want to publish.

It always leaves 0, 1, -1 numbers, dates, lengths of arrays, and null flags exactly as in source data. For example, you have a column `IsMobile` in your table with values 0 and 1. In transformed data, it will have the same value.

So, the user will be able to count the exact ratio of mobile traffic.

Let's give another example. When you have some private data in your table, like user email and you don't want to publish any single email address.

If your table is large enough and contains multiple different emails and no email has a very high frequency than all others, it will anonymize all data. But if you have a small number of different values in a column, it can reproduce some of them.

You should look at the working algorithm of this tool works, and fine-tune its command line parameters.

This tool works fine only with an average amount of data (at least 1000s of rows).

---

## clickhouse-odbc-bridge

Simple HTTP-server which works like a proxy for ODBC driver. The main motivation was possible segfaults or another faults in ODBC implementations, which can crash whole clickhouse-server process.

This tool works via HTTP, not via pipes, shared memory, or TCP because:

- It's simpler to implement
- It's simpler to debug
- jdbc-bridge can be implemented in the same way

## Usage

`clickhouse-server` use this tool inside odbc table function and StorageODBC.

However it can be used as standalone tool from command line with the following parameters in POST-request URL:

- `connection_string` -- ODBC connection string.
  - `columns` -- columns in ClickHouse NamesAndTypesList format, name in backticks, type as string. Name and type are space separated, rows separated with newline.
  - `max_block_size` -- optional parameter, sets maximum size of single block.
- Query is send in post body. Response is returned in RowBinary format.

## Example:

```
$ clickhouse-odbc-bridge --http-port 9018 --daemon

$ curl -d "query=SELECT PageID, ImplID, AdType FROM Keys ORDER BY PageID, ImplID" --data-urlencode
"connection_string=DSN=ClickHouse;DATABASE=stat" --data-urlencode "columns=columns format version: 1
3 columns:
\`PageID\` String
\`ImplID\` String
\`AdType\` String
" "http://localhost:9018/" > result.txt

$ cat result.txt
12246623837185725195925621517
```

## [experimental] OpenTelemetry Support

OpenTelemetry is an open standard for collecting traces and metrics from the distributed application. ClickHouse has some support for OpenTelemetry.

### Warning

This is an experimental feature that will change in backwards-incompatible ways in future releases.

## Supplying Trace Context to ClickHouse

ClickHouse accepts trace context HTTP headers, as described by the [W3C recommendation](#). It also accepts trace context over a native protocol that is used for communication between ClickHouse servers or between the client and server. For manual testing, trace context headers conforming to the Trace Context recommendation can be supplied to `clickhouse-client` using `--opentelemetry-traceparent` and `--opentelemetry-tracestate` flags.

If no parent trace context is supplied, ClickHouse can start a new trace, with probability controlled by the `opentelemetry_start_trace_probability` setting.

## Propagating the Trace Context

The trace context is propagated to downstream services in the following cases:

- Queries to remote ClickHouse servers, such as when using [Distributed](#) table engine.
- `url` table function. Trace context information is sent in HTTP headers.

## Tracing the ClickHouse Itself

ClickHouse creates trace spans for each query and some of the query execution stages, such as query planning or distributed queries.

To be useful, the tracing information has to be exported to a monitoring system that supports OpenTelemetry, such as [Jaeger](#) or [Prometheus](#). ClickHouse avoids a dependency on a particular monitoring system, instead only providing the tracing data through a system table. OpenTelemetry trace span information [required by the standard](#) is stored in the `system.opentelemetry_span_log` table.

The table must be enabled in the server configuration, see the `opentelemetry_span_log` element in the default config file `config.xml`. It is enabled by default.

The tags or attributes are saved as two parallel arrays, containing the keys and values. Use [ARRAY JOIN](#) to work with them.

# Integration with monitoring systems

At the moment, there is no ready tool that can export the tracing data from ClickHouse to a monitoring system.

For testing, it is possible to setup the export using a materialized view with the [URL](#) engine over the [system.opentelemetry\\_span\\_log](#) table, which would push the arriving log data to an HTTP endpoint of a trace collector. For example, to push the minimal span data to a Zipkin instance running at <http://localhost:9411>, in Zipkin v2 JSON format:

```
CREATE MATERIALIZED VIEW default.zipkin_spans
ENGINE = URL('http://127.0.0.1:9411/api/v2/spans', 'JSONEachRow')
SETTINGS output_format_json_named_tuples_as_objects = 1,
    output_format_json_array_of_rows = 1 AS
SELECT
    lower(hex(reinterpretAsFixedString(trace_id))) AS traceId,
    lower(hex(parent_span_id)) AS parentId,
    lower(hex(span_id)) AS id,
    operation_name AS name,
    start_time_us AS timestamp,
    finish_time_us - start_time_us AS duration,
    cast(tuple('clickhouse'), 'Tuple(serviceName text)') AS localEndpoint,
    cast(tuple(
        attribute.values[indexOf(attribute.names, 'db.statement')]),
        'Tuple("db.statement" text)') AS tags
FROM system.opentelemetry_span_log
```

In case of any errors, the part of the log data for which the error has occurred will be silently lost. Check the server log for error messages if the data does not arrive.

## Cache Types

When performing queries, ClickHouse uses different caches.

Main cache types:

- `mark_cache` — Cache of marks used by table engines of the [MergeTree](#) family.
- `uncompressed_cache` — Cache of uncompressed data used by table engines of the [MergeTree](#) family.

Additional cache types:

- DNS cache.
- [Regexp](#) cache.
- Compiled expressions cache.
- [Avro format](#) schemas cache.
- [Dictionaries](#) data cache.

Indirectly used:

- OS page cache.

To drop cache, use [SYSTEM DROP ... CACHE](#) statements.

## [pre-production] ClickHouse Keeper

ClickHouse server uses [ZooKeeper](#) coordination system for data [replication](#) and [distributed DDL](#) queries execution. ClickHouse Keeper is an alternative coordination system compatible with ZooKeeper.

## Warning

This feature is currently in the pre-production stage. We test it in our CI and on small internal installations.

## Implementation details

ZooKeeper is one of the first well-known open-source coordination systems. It's implemented in Java, has quite a simple and powerful data model. ZooKeeper's coordination algorithm called ZAB (ZooKeeper Atomic Broadcast) doesn't provide linearizability guarantees for reads, because each ZooKeeper node serves reads locally. Unlike ZooKeeper ClickHouse Keeper is written in C++ and uses [RAFT algorithm implementation](#). This algorithm allows to have linearizability for reads and writes, has several open-source implementations in different languages.

By default, ClickHouse Keeper provides the same guarantees as ZooKeeper (linearizable writes, non-linearizable reads). It has a compatible client-server protocol, so any standard ZooKeeper client can be used to interact with ClickHouse Keeper. Snapshots and logs have an incompatible format with ZooKeeper, but `clickhouse-keeper-converter` tool allows to convert ZooKeeper data to ClickHouse Keeper snapshot. Interserver protocol in ClickHouse Keeper is also incompatible with ZooKeeper so mixed ZooKeeper / ClickHouse Keeper cluster is impossible.

## Configuration

ClickHouse Keeper can be used as a standalone replacement for ZooKeeper or as an internal part of the ClickHouse server, but in both cases configuration is almost the same .xml file. The main ClickHouse Keeper configuration tag is `<keeper_server>`. Keeper configuration has the following parameters:

- `tcp_port` — Port for a client to connect (default for ZooKeeper is 2181).
- `tcp_port_secure` — Secure port for a client to connect.
- `server_id` — Unique server id, each participant of the ClickHouse Keeper cluster must have a unique number (1, 2, 3, and so on).
- `log_storage_path` — Path to coordination logs, better to store logs on the non-busy device (same for ZooKeeper).
- `snapshot_storage_path` — Path to coordination snapshots.

Other common parameters are inherited from the ClickHouse server config (`listen_host`, `logger`, and so on).

Internal coordination settings are located in `<keeper_server>. <coordination_settings>` section:

- `operation_timeout_ms` — Timeout for a single client operation (ms) (default: 10000).
- `session_timeout_ms` — Timeout for client session (ms) (default: 30000).
- `dead_session_check_period_ms` — How often ClickHouse Keeper check dead sessions and remove them (ms) (default: 500).
- `heart_beat_interval_ms` — How often a ClickHouse Keeper leader will send heartbeats to followers (ms) (default: 500).
- `election_timeout_lower_bound_ms` — If the follower didn't receive heartbeats from the leader in this interval, then it can initiate leader election (default: 1000).

- `election_timeout_upper_bound_ms` — If the follower didn't receive heartbeats from the leader in this interval, then it must initiate leader election (default: 2000).
- `rotate_log_storage_interval` — How many log records to store in a single file (default: 100000).
- `reserved_log_items` — How many coordination log records to store before compaction (default: 100000).
- `snapshot_distance` — How often ClickHouse Keeper will create new snapshots (in the number of records in logs) (default: 100000).
- `snapshots_to_keep` — How many snapshots to keep (default: 3).
- `stale_log_gap` — Threshold when leader considers follower as stale and sends the snapshot to it instead of logs (default: 10000).
- `fresh_log_gap` — When node became fresh (default: 200).
- `max_requests_batch_size` - Max size of batch in requests count before it will be sent to RAFT (default: 100).
- `force_sync` — Call `fsync` on each write to coordination log (default: true).
- `quorum_reads` — Execute read requests as writes through whole RAFT consensus with similar speed (default: false).
- `raft_logs_level` — Text logging level about coordination (trace, debug, and so on) (default: system default).
- `auto_forwarding` — Allow to forward write requests from followers to the leader (default: true).
- `shutdown_timeout` — Wait to finish internal connections and shutdown (ms) (default: 5000).
- `startup_timeout` — If the server doesn't connect to other quorum participants in the specified timeout it will terminate (ms) (default: 30000).

Quorum configuration is located in `<keeper_server>.<raft_configuration>` section and contain servers description. The only parameter for the whole quorum is `secure`, which enables encrypted connection for communication between quorum participants. The main parameters for each `<server>` are:

- `id` — Server identifier in a quorum.
- `hostname` — Hostname where this server is placed.
- `port` — Port where this server listens for connections.

Examples of configuration for quorum with three nodes can be found in [integration tests](#) with `test_keeper_` prefix. Example configuration for server #1:

```

<keeper_server>
  <tcp_port>2181</tcp_port>
  <server_id>1</server_id>
  <log_storage_path>/var/lib/clickhouse/coordination/log</log_storage_path>
  <snapshot_storage_path>/var/lib/clickhouse/coordination/snapshots</snapshot_storage_path>

  <coordination_settings>
    <operation_timeout_ms>10000</operation_timeout_ms>
    <session_timeout_ms>30000</session_timeout_ms>
    <raft_logs_level>trace</raft_logs_level>
  </coordination_settings>

  <raft_configuration>
    <server>
      <id>1</id>
      <hostname>zoo1</hostname>
      <port>9444</port>
    </server>
    <server>
      <id>2</id>
      <hostname>zoo2</hostname>
      <port>9444</port>
    </server>
    <server>
      <id>3</id>
      <hostname>zoo3</hostname>
      <port>9444</port>
    </server>
  </raft_configuration>
</keeper_server>

```

## How to run

ClickHouse Keeper is bundled into the ClickHouse server package, just add configuration of `<keeper_server>` and start ClickHouse server as always. If you want to run standalone ClickHouse Keeper you can start it in a similar way with:

```
clickhouse-keeper --config /etc/your_path_to_config/config.xml --daemon
```

## [experimental] Migration from ZooKeeper

Seamlessly migration from ZooKeeper to ClickHouse Keeper is impossible you have to stop your ZooKeeper cluster, convert data and start ClickHouse Keeper. `clickhouse-keeper-converter` tool allows converting ZooKeeper logs and snapshots to ClickHouse Keeper snapshot. It works only with ZooKeeper > 3.4. Steps for migration:

1. Stop all ZooKeeper nodes.
2. Optional, but recommended: find ZooKeeper leader node, start and stop it again. It will force ZooKeeper to create a consistent snapshot.
3. Run `clickhouse-keeper-converter` on a leader, for example:

```
clickhouse-keeper-converter --zookeeper-logs-dir /var/lib/zookeeper/version-2 --zookeeper-snapshots-dir /var/lib/zookeeper/version-2 --output-dir /path/to/clickhouse/keeper/snapshots
```

4. Copy snapshot to ClickHouse server nodes with a configured `keeper` or start ClickHouse Keeper instead of ZooKeeper. The snapshot must persist on all nodes, otherwise, empty nodes can be faster and one of them can become a leader.

## External Disks for Storing Data

Data, processed in ClickHouse, is usually stored in the local file system — on the same machine with the ClickHouse server. That requires large-capacity disks, which can be expensive enough. To avoid that you can store the data remotely — on [Amazon S3](#) disks or in the Hadoop Distributed File System ([HDFS](#)).

To work with data stored on [Amazon S3](#) disks use [S3](#) table engine, and to work with data in the Hadoop Distributed File System — [HDFS](#) table engine.

To load data from a web server with static files use a disk with type [web](#).

## Zero-copy Replication

ClickHouse supports zero-copy replication for [S3](#) and [HDFS](#) disks, which means that if the data is stored remotely on several machines and needs to be synchronized, then only the metadata is replicated (paths to the data parts), but not the data itself.

## Configuring HDFS

[MergeTree](#) and [Log](#) family table engines can store data to HDFS using a disk with type [HDFS](#).

Configuration markup:

```
<clickhouse>
  <storage_configuration>
    <disks>
      <hdfs>
        <type>hdfs</type>
        <endpoint>hdfs://hdfs1:9000/clickhouse/</endpoint>
      </hdfs>
    </disks>
    <policies>
      <hdfs>
        <volumes>
          <main>
            <disk>hdfs</disk>
          </main>
        </volumes>
      </hdfs>
    </policies>
  </storage_configuration>

  <merge_tree>
    <min_bytes_for_wide_part>0</min_bytes_for_wide_part>
  </merge_tree>
</clickhouse>
```

Required parameters:

- `endpoint` — HDFS endpoint URL in `path` format. Endpoint URL should contain a root path to store data.

Optional parameters:

- `min_bytes_for_seek` — The minimal number of bytes to use seek operation instead of sequential read.  
Default value: `1 Mb`.

## Using Virtual File System for Data Encryption

You can encrypt the data stored on [S3](#), or [HDFS](#) external disks, or on a local disk. To turn on the encryption mode, in the configuration file you must define a disk with the type `encrypted` and choose a disk on which the data will be saved. An `encrypted` disk ciphers all written files on the fly, and when you read files from an `encrypted` disk it deciphers them automatically. So you can work with an `encrypted` disk like with a normal one.

Example of disk configuration:

```

<disks>
  <disk1>
    <type>local</type>
    <path>/path1/</path>
  </disk1>
  <disk2>
    <type>encrypted</type>
    <disk>disk1</disk>
    <path>path2/</path>
    <key>_16_ascii_chars_</key>
  </disk2>
</disks>

```

For example, when ClickHouse writes data from some table to a file `store/all_1_1_0/data.bin` to `disk1`, then in fact this file will be written to the physical disk along the path `/path1/store/all_1_1_0/data.bin`.

When writing the same file to `disk2`, it will actually be written to the physical disk at the path `/path1/path2/store/all_1_1_0/data.bin` in encrypted mode.

Required parameters:

- `type` — encrypted. Otherwise the encrypted disk is not created.
- `disk` — Type of disk for data storage.
- `key` — The key for encryption and decryption. Type: [UInt64](#). You can use `key_hex` parameter to encrypt in hexadecimal form.  
You can specify multiple keys using the `id` attribute (see example above).

Optional parameters:

- `path` — Path to the location on the disk where the data will be saved. If not specified, the data will be saved in the root directory.
- `current_key_id` — The key used for encryption. All the specified keys can be used for decryption, and you can always switch to another key while maintaining access to previously encrypted data.
- `algorithm` — [Algorithm](#) for encryption. Possible values: `AES_128_CTR`, `AES_192_CTR` or `AES_256_CTR`. Default value: `AES_128_CTR`. The key length depends on the algorithm: `AES_128_CTR` — 16 bytes, `AES_192_CTR` — 24 bytes, `AES_256_CTR` — 32 bytes.

Example of disk configuration:

```

<clickhouse>
  <storage_configuration>
    <disks>
      <disk_s3>
        <type>s3</type>
        <endpoint>...
      </disk_s3>
      <disk_s3_encrypted>
        <type>encrypted</type>
        <disk>disk_s3</disk>
        <algorithm>AES_128_CTR</algorithm>
        <key_hex id="0">00112233445566778899aabccddeeff</key_hex>
        <key_hex id="1">ffeedddccbba99887766554433221100</key_hex>
        <current_key_id>1</current_key_id>
      </disk_s3_encrypted>
    </disks>
  </storage_configuration>
</clickhouse>

```

## Storing Data on Web Server

There is a tool `clickhouse-static-files-uploader`, which prepares a data directory for a given table (`SELECT data_paths FROM system.tables WHERE name = 'table_name'`). For each table you need, you get a directory of files. These files can be uploaded to, for example, a web server with static files. After this preparation, you can load this table into any ClickHouse server via DiskWeb.

This is a read-only disk. Its data is only read and never modified. A new table is loaded to this disk via `ATTACH TABLE` query (see example below). Local disk is not actually used, each `SELECT` query will result in a http request to fetch required data. All modification of the table data will result in an exception, i.e. the following types of queries are not allowed: `CREATE TABLE`, `ALTER TABLE`, `RENAME TABLE`, `DETACH TABLE` and `TRUNCATE TABLE`.

Web server storage is supported only for the `MergeTree` and `Log` engine families. To access the data stored on a web disk, use the `storage_policy` setting when executing the query. For example, `ATTACH TABLE table_web UUID '{}' (id Int32) ENGINE = MergeTree() ORDER BY id SETTINGS storage_policy = 'web'`.

A ready test case. You need to add this configuration to config:

```
<clickhouse>
  <storage_configuration>
    <disks>
      <web>
        <type>web</type>
        <endpoint>https://clickhouse-datasets.s3.yandex.net/disk-with-static-files-tests/test-hits/</endpoint>
      </web>
    </disks>
    <policies>
      <web>
        <volumes>
          <main>
            <disk>web</disk>
          </main>
        </volumes>
      </web>
    </policies>
  </storage_configuration>
</clickhouse>
```

And then execute this query:

```
ATTACH TABLE test_hits UUID '1ae36516-d62d-4218-9ae3-6516d62da218'
(
    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
```

```
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,
NSToDOMContentLoadedTiming Int32,
FirstPaintTiming Int32,
RedirectCount Int8,
SocialSourceNetworkID UInt8,
SocialSourcePage String,
ParamPrice Int64,
ParamOrderID String,
ParamCurrency FixedString(3),
ParamCurrencyID UInt16
```

```

ParanCurrencyID 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 storage_policy='web';

```

Required parameters:

- `type` — `web`. Otherwise the disk is not created.
- `endpoint` — The endpoint URL in `path` format. Endpoint URL must contain a root path to store data, where they were uploaded.

Optional parameters:

- `min_bytes_for_seek` — The minimal number of bytes to use seek operation instead of sequential read. Default value: `1 Mb`.
- `remote_fs_read_backoff_threshold` — The maximum wait time when trying to read data for remote disk. Default value: `10000` seconds.
- `remote_fs_read_backoff_max_tries` — The maximum number of attempts to read with backoff. Default value: `5`.

If a query fails with an exception `DB:Exception Unreachable URL`, then you can try to adjust the settings: `http_connection_timeout`, `http_receive_timeout`, `keep_alive_timeout`.

To get files for upload run:

```
clickhouse static-files-disk-uploader --metadata-path <path> --output-dir <dir> (--metadata-path can be found in query SELECT data_paths FROM system.tables WHERE name = 'table_name').
```

When loading files by `endpoint`, they must be loaded into `<endpoint>/store/` path, but config must contain only `endpoint`.

If URL is not reachable on disk load when the server is starting up tables, then all errors are caught. If in this case there were errors, tables can be reloaded (become visible) via `DETACH TABLE table_name -> ATTACH TABLE table_name`. If metadata was successfully loaded at server startup, then tables are available straight away.

Use `http_max_single_read_retries` setting to limit the maximum number of retries during a single HTTP read.

---

toc\_priority: 58  
toc\_title: Usage Recommendations

---

## ClickHouse Development

---

### The Beginner ClickHouse Developer Instruction

Building of ClickHouse is supported on Linux, FreeBSD and Mac OS X.

If you use Windows, you need to create a virtual machine with Ubuntu. To start working with a virtual machine please install VirtualBox. You can download Ubuntu from the website:

<https://www.ubuntu.com/#download>. Please create a virtual machine from the downloaded image (you should reserve at least 4GB of RAM for it). To run a command-line terminal in Ubuntu, please locate a program containing the word “terminal” in its name (gnome-terminal, konsole etc.) or just press Ctrl+Alt+T.

ClickHouse cannot work or build on a 32-bit system. You should acquire access to a 64-bit system and you can continue reading.

### Creating a Repository on GitHub

To start working with ClickHouse repository you will need a GitHub account.

You probably already have one, but if you do not, please register at <https://github.com>. In case you do not have SSH keys, you should generate them and then upload them on GitHub. It is required for sending over your patches. It is also possible to use the same SSH keys that you use with any other SSH servers - probably you already have those.

Create a fork of ClickHouse repository. To do that please click on the “fork” button in the upper right corner at <https://github.com/ClickHouse/ClickHouse>. It will fork your own copy of ClickHouse/ClickHouse to your account.

The development process consists of first committing the intended changes into your fork of ClickHouse and then creating a “pull request” for these changes to be accepted into the main repository (ClickHouse/ClickHouse).

To work with git repositories, please install git.

To do that in Ubuntu you would run in the command line terminal:

```
sudo apt update
sudo apt install git
```

A brief manual on using Git can be found here: <https://education.github.com/git-cheat-sheet-education.pdf>. For a detailed manual on Git see <https://git-scm.com/book/en/v2>.

# Cloning a Repository to Your Development Machine

Next, you need to download the source files onto your working machine. This is called “to clone a repository” because it creates a local copy of the repository on your working machine.

In the command line terminal run:

```
git clone git@github.com:your_github_username/ClickHouse.git  
cd ClickHouse
```

Note: please, substitute *your\_github\_username* with what is appropriate!

This command will create a directory `ClickHouse` containing the working copy of the project.

It is important that the path to the working directory contains no whitespaces as it may lead to problems with running the build system.

Please note that ClickHouse repository uses submodules. That is what the references to additional repositories are called (i.e. external libraries on which the project depends). It means that when cloning the repository you need to specify the `--recursive` flag as in the example above. If the repository has been cloned without submodules, to download them you need to run the following:

```
git submodule init  
git submodule update
```

You can check the status with the command: `git submodule status`.

If you get the following error message:

```
Permission denied (publickey).  
fatal: Could not read from remote repository.
```

```
Please make sure you have the correct access rights  
and the repository exists.
```

It generally means that the SSH keys for connecting to GitHub are missing. These keys are normally located in `~/.ssh`. For SSH keys to be accepted you need to upload them in the settings section of GitHub UI.

You can also clone the repository via https protocol:

```
git clone https://github.com/ClickHouse/ClickHouse.git
```

This, however, will not let you send your changes to the server. You can still use it temporarily and add the SSH keys later replacing the remote address of the repository with `git remote` command.

You can also add original ClickHouse repo’s address to your local repository to pull updates from there:

```
git remote add upstream git@github.com:ClickHouse/ClickHouse.git
```

After successfully running this command you will be able to pull updates from the main ClickHouse repo by running `git pull upstream master`.

## Working with Submodules

Working with submodules in git could be painful. Next commands will help to manage it:

```
# ! each command accepts
# 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
```

The next commands would help you to reset all submodules to the initial state (!WARNING! - any changes inside will be deleted):

```
# Synchronizes submodules' remote URL with .gitmodules
git submodule sync
# Update the registered submodules with initialize not yet initialized
git submodule update --init
# 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
git submodule foreach git submodule update --init
git submodule foreach git submodule foreach git reset --hard
git submodule foreach git submodule foreach git clean -xfd
```

## Build System

ClickHouse uses CMake and Ninja for building.

CMake - a meta-build system that can generate Ninja files (build tasks).

Ninja - a smaller build system with a focus on the speed used to execute those cmake generated tasks.

To install on Ubuntu, Debian or Mint run `sudo apt install cmake ninja-build`.

On CentOS, RedHat run `sudo yum install cmake ninja-build`.

If you use Arch or Gentoo, you probably know it yourself how to install CMake.

For installing CMake and Ninja on Mac OS X first install Homebrew and then install everything else via brew:

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
brew install cmake ninja
```

Next, check the version of CMake: `cmake --version`. If it is below 3.12, you should install a newer version from the website: <https://cmake.org/download/>.

## Optional External Libraries

ClickHouse uses several external libraries for building. All of them do not need to be installed separately as they are built together with ClickHouse from the sources located in the submodules. You can check the list in `contrib`.

## C++ Compiler

Compilers Clang starting from version 11 is supported for building ClickHouse.

Clang should be used instead of gcc. Though, our continuous integration (CI) platform runs checks for about a dozen of build combinations.

On Ubuntu/Debian you can use the automatic installation script (check [official webpage](#))

```
sudo bash -c "$(wget -O - https://apt.llvm.org/llvm.sh)"
```

Mac OS X build is also supported. Just run brew install llvm

## The Building Process

Now that you are ready to build ClickHouse we recommend you to create a separate directory build inside ClickHouse that will contain all of the build artefacts:

```
mkdir build  
cd build
```

You can have several different directories (build\_release, build\_debug, etc.) for different types of build.

While inside the `build` directory, configure your build by running CMake. Before the first run, you need to define environment variables that specify compiler.

```
export CC=clang CXX=clang++  
cmake ..
```

The `CC` variable specifies the compiler for C (short for C Compiler), and `CXX` variable instructs which C++ compiler is to be used for building.

For a faster build, you can resort to the `debug` build type - a build with no optimizations. For that supply the following parameter `-D CMAKE_BUILD_TYPE=Debug`:

```
cmake -D CMAKE_BUILD_TYPE=Debug ..
```

You can change the type of build by running this command in the `build` directory.

Run `ninja` to build:

```
ninja clickhouse-server clickhouse-client
```

Only the required binaries are going to be built in this example.

If you require to build all the binaries (utilities and tests), you should run `ninja` with no parameters:

```
ninja
```

Full build requires about 30GB of free disk space or 15GB to build the main binaries.

When a large amount of RAM is available on build machine you should limit the number of build tasks run in parallel with `-j` param:

```
ninja -j 1 clickhouse-server clickhouse-client
```

On machines with 4GB of RAM, it is recommended to specify 1, for 8GB of RAM `-j 2` is recommended.

If you get the message: `ninja: error: loading 'build.ninja': No such file or directory`, it means that generating a build configuration has failed and you need to inspect the message above.

Upon the successful start of the building process, you'll see the build progress - the number of processed tasks and the total number of tasks.

While building messages about protobuf files in libhdfs2 library like `libprotobuf` WARNING may show up. They affect nothing and are safe to be ignored.

Upon successful build you get an executable file `ClickHouse/<build_dir>/programs/clickhouse`:

```
ls -l programs/clickhouse
```

## Running the Built Executable of ClickHouse

To run the server under the current user you need to navigate to `ClickHouse/programs/server/` (located outside of build) and run:

```
../../build/programs/clickhouse server
```

In this case, ClickHouse will use config files located in the current directory. You can run `clickhouse` server from any directory specifying the path to a config file as a command-line parameter `--config-file`.

To connect to ClickHouse with `clickhouse-client` in another terminal navigate to `ClickHouse/build/programs/` and run `./clickhouse client`.

If you get `Connection refused` message on Mac OS X or FreeBSD, try specifying host address `127.0.0.1`:

```
clickhouse client --host 127.0.0.1
```

You can replace the production version of ClickHouse binary installed in your system with your custom-built ClickHouse binary. To do that install ClickHouse on your machine following the instructions from the official website. Next, run the following:

```
sudo service clickhouse-server stop
sudo cp ClickHouse/build/programs/clickhouse /usr/bin/
sudo service clickhouse-server start
```

Note that `clickhouse-client`, `clickhouse-server` and others are symlinks to the commonly shared `clickhouse` binary.

You can also run your custom-built ClickHouse binary with the config file from the ClickHouse package installed on your system:

```
sudo service clickhouse-server stop
sudo -u clickhouse ClickHouse/build/programs/clickhouse server --config-file /etc/clickhouse-server/config.xml
```

## IDE (Integrated Development Environment)

If you do not know which IDE to use, we recommend that you use CLion. CLion is commercial software, but it offers 30 days free trial period. It is also free of charge for students. CLion can be used both on Linux and on Mac OS X.

KDevelop and QTCreator are other great alternatives of an IDE for developing ClickHouse. KDevelop comes in as a very handy IDE although unstable. If KDevelop crashes after a while upon opening project, you should click "Stop All" button as soon as it has opened the list of project's files. After doing so KDevelop should be fine to work with.

As simple code editors, you can use Sublime Text or Visual Studio Code, or Kate (all of which are available on Linux).

Just in case, it is worth mentioning that CLion creates build path on its own, it also on its own selects debug for build type, for configuration it uses a version of CMake that is defined in CLion and not the one installed by you, and finally, CLion will use `make` to run build tasks instead of `ninja`. This is normal behaviour, just keep that in mind to avoid confusion.

## Writing Code

The description of ClickHouse architecture can be found here:

<https://clickhouse.com/docs/en/development/architecture/>

The Code Style Guide: <https://clickhouse.com/docs/en/development/style/>

Adding third-party libraries: <https://clickhouse.com/docs/en/development/contrib/#adding-third-party-libraries>

Writing tests: <https://clickhouse.com/docs/en/development/tests/>

List of tasks: <https://github.com/ClickHouse/ClickHouse/issues?q=is%3Aopen+is%3Aissue+label%3A%22easy+task%22>

## Test Data

Developing ClickHouse often requires loading realistic datasets. It is particularly important for performance testing. We have a specially prepared set of anonymized data from Yandex.Metrica. It requires additionally some 3GB of free disk space. Note that this data is not required to accomplish most of the development tasks.

```
sudo apt install wget xz-utils

wget https://datasets.clickhouse.com/hits/tsv/hits_v1.tsv.xz
wget https://datasets.clickhouse.com/visits/tsv/visits_v1.tsv.xz

xz -v -d hits_v1.tsv.xz
xz -v -d visits_v1.tsv.xz

clickhouse-client

CREATE DATABASE IF NOT EXISTS test

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, 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.Key1` Array(String), `ParsedParams.Key2` Array(String), `ParsedParams.Key3` Array(String), `ParsedParams.Key4` 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
UInt32, RefererCategories Array(UInt16), URLCategories Array(UInt16), URLRegions Array(UInt32), RefererRegions
Array(UInt32), IsYandex UInt8, GoalReachesDepth UInt32, GoalReachesURL UInt32, GoalReachesAny UInt32,
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 UInt32, ClickGoodEvent
UInt32, ClickEventTime DateTime, ClickPriorityID UInt32, ClickPhraseID UInt32, ClickPageID UInt32, ClickPlaceID UInt32,
ClickTypeID UInt32, ClickResourceID UInt32, 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 UInt8, ClickOptions String, ClickGroupBannerID UInt32, OpenstatServiceName String,
OpenstatCampaignID String, OpenstatAdID String, OpenstatSourceID String, UTMSource String, UTMMedium String,
UTMCampaign String, UTMCContent 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

```

## Creating Pull Request

Navigate to your fork repository in GitHub's UI. If you have been developing in a branch, you need to select that branch. There will be a “Pull request” button located on the screen. In essence, this means “create a request for accepting my changes into the main repository”.

A pull request can be created even if the work is not completed yet. In this case please put the word “WIP” (work in progress) at the beginning of the title, it can be changed later. This is useful for cooperative reviewing and discussion of changes as well as for running all of the available tests. It is important that you provide a brief description of your changes, it will later be used for generating release changelogs.

Testing will commence as soon as Yandex employees label your PR with a tag “can be tested”. The results of some first checks (e.g. code style) will come in within several minutes. Build check results will arrive within half an hour. And the main set of tests will report itself within an hour.

The system will prepare ClickHouse binary builds for your pull request individually. To retrieve these builds click the “Details” link next to “ClickHouse build check” entry in the list of checks. There you will find direct links to the built .deb packages of ClickHouse which you can deploy even on your production servers (if you have no fear).

Most probably some of the builds will fail at first times. This is due to the fact that we check builds both with gcc as well as with clang, with almost all of existing warnings (always with the `-Werror` flag) enabled for clang. On that same page, you can find all of the build logs so that you do not have to build ClickHouse in all of the possible ways.

## Overview of ClickHouse Architecture

ClickHouse is a true column-oriented DBMS. Data is stored by columns, and during the execution of arrays (vectors or chunks of columns). Whenever possible, operations are dispatched on arrays, rather than on individual values. It is called “vectorized query execution” and it helps lower the cost of actual data processing.

This idea is nothing new. It dates back to the APL (A programming language, 1957) and its descendants: A + (APL dialect), J (1990), K (1993), and Q (programming language from Kx Systems, 2003). Array programming is used in scientific data processing. Neither is this idea something new in relational databases: for example, it is used in the VectorWise system (also known as Actian Vector Analytic Database by Actian Corporation).

There are two different approaches for speeding up query processing: vectorized query execution and runtime code generation. The latter removes all indirection and dynamic dispatch. Neither of these approaches is strictly better than the other. Runtime code generation can be better when it fuses many operations, thus fully utilizing CPU execution units and the pipeline. Vectorized query execution can be less practical because it involves temporary vectors that must be written to the cache and read back. If the temporary data does not fit in the L2 cache, this becomes an issue. But vectorized query execution more easily utilizes the SIMD capabilities of the CPU. A [research paper](#) written by our friends shows that it is better to combine both approaches. ClickHouse uses vectorized query execution and has limited initial support for runtime code generation.

## Columns

`IColumn` interface is used to represent columns in memory (actually, chunks of columns). This interface provides helper methods for the implementation of various relational operators. Almost all operations are immutable: they do not modify the original column, but create a new modified one. For example, the `IColumn :: filter` method accepts a filter byte mask. It is used for the `WHERE` and `HAVING` relational operators. Additional examples: the `IColumn :: permute` method to support `ORDER BY`, the `IColumn :: cut` method to support `LIMIT`.

Various `IColumn` implementations (`ColumnUInt8`, `ColumnString`, and so on) are responsible for the memory layout of columns. The memory layout is usually a contiguous array. For the integer type of columns, it is just one contiguous array, like `std :: vector`. For `String` and `Array` columns, it is two vectors: one for all array elements, placed contiguously, and a second one for offsets to the beginning of each array. There is also `ColumnConst` that stores just one value in memory, but looks like a column.

## Field

Nevertheless, it is possible to work with individual values as well. To represent an individual value, the `Field` is used. `Field` is just a discriminated union of `UInt64`, `Int64`, `Float64`, `String` and `Array`. `IColumn` has the operator `[]` method to get the n-th value as a `Field`, and the `insert` method to append a `Field` to the end of a column.

These methods are not very efficient, because they require dealing with temporary `Field` objects representing an individual value. There are more efficient methods, such as `insertFrom`, `insertRangeFrom`, and so on.

`Field` does not have enough information about a specific data type for a table. For example, `UInt8`, `UInt16`, `UInt32`, and `UInt64` are all represented as `UInt64` in a `Field`.

## Leaky Abstractions

`IColumn` has methods for common relational transformations of data, but they do not meet all needs. For example, `ColumnUInt64` does not have a method to calculate the sum of two columns, and `ColumnString` does not have a method to run a substring search. These countless routines are implemented outside of `IColumn`.

Various functions on columns can be implemented in a generic, non-efficient way using `IColumn` methods to extract `Field` values, or in a specialized way using knowledge of inner memory layout of data in a specific `IColumn` implementation. It is implemented by casting functions to a specific `IColumn` type and deal with internal representation directly. For example, `ColumnUInt64` has the `getData` method that returns a reference to an internal array, then a separate routine reads or fills that array directly. We have “leaky abstractions” to allow efficient specializations of various routines.

## Data Types

`IDataType` is responsible for serialization and deserialization: for reading and writing chunks of columns or individual values in binary or text form. `IDataType` directly corresponds to data types in tables. For example, there are `DataTypeUInt32`, `DataTypeDateTime`, `DataTypeString` and so on.

`IDataType` and `IColumn` are only loosely related to each other. Different data types can be represented in memory by the same `IColumn` implementations. For example, `DataTypeUInt32` and `DataTypeDateTime` are both represented by `ColumnUInt32` or `ColumnConstUInt32`. In addition, the same data type can be represented by different `IColumn` implementations. For example, `DataTypeUInt8` can be represented by `ColumnUInt8` or `ColumnConstUInt8`.

`IDataType` only stores metadata. For instance, `DataTypeUInt8` does not store anything at all (except virtual pointer `vptr`) and `DataTypeFixedString` stores just `N` (the size of fixed-size strings).

`IDataType` has helper methods for various data formats. Examples are methods to serialize a value with possible quoting, to serialize a value for JSON, and to serialize a value as part of the XML format. There is no direct correspondence to data formats. For example, the different data formats `Pretty` and `TabSeparated` can use the same `serializeTextEscaped` helper method from the `IDataType` interface.

## Block

A `Block` is a container that represents a subset (chunk) of a table in memory. It is just a set of triples: (`IColumn`, `IDataType`, column name). During query execution, data is processed by `Blocks`. If we have a `Block`, we have data (in the `IColumn` object), we have information about its type (in `IDataType`) that tells us how to deal with that column, and we have the column name. It could be either the original column name from the table or some artificial name assigned for getting temporary results of calculations.

When we calculate some function over columns in a block, we add another column with its result to the block, and we do not touch columns for arguments of the function because operations are immutable. Later, unneeded columns can be removed from the block, but not modified. It is convenient for the elimination of common subexpressions.

Blocks are created for every processed chunk of data. Note that for the same type of calculation, the column names and types remain the same for different blocks, and only column data changes. It is better to split block data from the block header because small block sizes have a high overhead of temporary strings for copying `shared_ptrs` and column names.

# Block Streams

Block streams are for processing data. We use streams of blocks to read data from somewhere, perform data transformations, or write data to somewhere. `IBlockInputStream` has the `read` method to fetch the next block while available. `IBlockOutputStream` has the `write` method to push the block somewhere.

Streams are responsible for:

1. Reading or writing to a table. The table just returns a stream for reading or writing blocks.
2. Implementing data formats. For example, if you want to output data to a terminal in `Pretty` format, you create a block output stream where you push blocks, and it formats them.
3. Performing data transformations. Let's say you have `IBlockInputStream` and want to create a filtered stream. You create `FilterBlockInputStream` and initialize it with your stream. Then when you pull a block from `FilterBlockInputStream`, it pulls a block from your stream, filters it, and returns the filtered block to you. Query execution pipelines are represented this way.

There are more sophisticated transformations. For example, when you pull from `AggregatingBlockInputStream`, it reads all data from its source, aggregates it, and then returns a stream of aggregated data for you.

Another example: `UnionBlockInputStream` accepts many input sources in the constructor and also a number of threads. It launches multiple threads and reads from multiple sources in parallel.

Block streams use the “pull” approach to control flow: when you pull a block from the first stream, it consequently pulls the required blocks from nested streams, and the entire execution pipeline will work. Neither “pull” nor “push” is the best solution, because control flow is implicit, and that limits the implementation of various features like simultaneous execution of multiple queries (merging many pipelines together). This limitation could be overcome with coroutines or just running extra threads that wait for each other. We may have more possibilities if we make control flow explicit: if we locate the logic for passing data from one calculation unit to another outside of those calculation units. Read this [article](#) for more thoughts.

We should note that the query execution pipeline creates temporary data at each step. We try to keep block size small enough so that temporary data fits in the CPU cache. With that assumption, writing and reading temporary data is almost free in comparison with other calculations. We could consider an alternative, which is to fuse many operations in the pipeline together. It could make the pipeline as short as possible and remove much of the temporary data, which could be an advantage, but it also has drawbacks. For example, a split pipeline makes it easy to implement caching intermediate data, stealing intermediate data from similar queries running at the same time, and merging pipelines for similar queries.

## Formats

Data formats are implemented with block streams. There are “presentational” formats only suitable for the output of data to the client, such as `Pretty` format, which provides only `IBlockOutputStream`. And there are input/output formats, such as `TabSeparated` or `JSONEachRow`.

There are also row streams: `IRowInputStream` and `IRowOutputStream`. They allow you to pull/push data by individual rows, not by blocks. And they are only needed to simplify the implementation of row-oriented formats. The wrappers `BlockInputStreamFromRowInputStream` and `BlockOutputStreamFromRowOutputStream` allow you to convert row-oriented streams to regular block-oriented streams.

## I/O

For byte-oriented input/output, there are `ReadBuffer` and `WriteBuffer` abstract classes. They are used instead of C++ `iostreams`. Don't worry: every mature C++ project is using something other than `iostreams` for good reasons.

`ReadBuffer` and `WriteBuffer` are just a contiguous buffer and a cursor pointing to the position in that buffer. Implementations may own or not own the memory for the buffer. There is a virtual method to fill the buffer with the following data (for `ReadBuffer`) or to flush the buffer somewhere (for `WriteBuffer`). The virtual methods are rarely called.

Implementations of `ReadBuffer/WriteBuffer` are used for working with files and file descriptors and network sockets, for implementing compression (`CompressedWriteBuffer` is initialized with another `WriteBuffer` and performs compression before writing data to it), and for other purposes – the names `ConcatReadBuffer`, `LimitReadBuffer`, and `HashingWriteBuffer` speak for themselves.

`Read/WriteBuffers` only deal with bytes. There are functions from `ReadHelpers` and `WriteHelpers` header files to help with formatting input/output. For example, there are helpers to write a number in decimal format.

Let's look at what happens when you want to write a result set in `JSON` format to `stdout`. You have a result set ready to be fetched from `IBlockInputStream`. You create `WriteBufferFromFileDescriptor(STDOUT_FILENO)` to write bytes to `stdout`. You create `JSONRowOutputStream`, initialized with that `WriteBuffer`, to write rows in `JSON` to `stdout`. You create `BlockOutputStreamFromRowOutputStream` on top of it, to represent it as `IBlockOutputStream`. Then you call `copyData` to transfer data from `IBlockInputStream` to `IBlockOutputStream`, and everything works. Internally, `JSONRowOutputStream` will write various `JSON` delimiters and call the `IDataType::serializeTextJSON` method with a reference to `IColumn` and the row number as arguments. Consequently, `IDataType::serializeTextJSON` will call a method from `WriteHelpers.h`: for example, `writeText` for numeric types and `writeJSONString` for `DataTypeString`.

## Tables

The `IStorage` interface represents tables. Different implementations of that interface are different table engines. Examples are `StorageMergeTree`, `StorageMemory`, and so on. Instances of these classes are just tables.

The key `IStorage` methods are `read` and `write`. There are also `alter`, `rename`, `drop`, and so on. The `read` method accepts the following arguments: the set of columns to read from a table, the `AST` query to consider, and the desired number of streams to return. It returns one or multiple `IBlockInputStream` objects and information about the stage of data processing that was completed inside a table engine during query execution.

In most cases, the `read` method is only responsible for reading the specified columns from a table, not for any further data processing. All further data processing is done by the query interpreter and is outside the responsibility of `IStorage`.

But there are notable exceptions:

- The `AST` query is passed to the `read` method, and the table engine can use it to derive index usage and to read fewer data from a table.
- Sometimes the table engine can process data itself to a specific stage. For example, `StorageDistributed` can send a query to remote servers, ask them to process data to a stage where data from different remote servers can be merged, and return that preprocessed data. The query interpreter then finishes processing the data.

The table's `read` method can return multiple `IBlockInputStream` objects to allow parallel data processing. These multiple block input streams can read from a table in parallel. Then you can wrap these streams with various transformations (such as expression evaluation or filtering) that can be calculated independently and create a `UnionBlockInputStream` on top of them, to read from multiple streams in parallel.

There are also `TableFunctions`. These are functions that return a temporary `IStorage` object to use in the `FROM` clause of a query.

To get a quick idea of how to implement your table engine, look at something simple, like `StorageMemory` or `StorageTinyLog`.

As the result of the `read` method, `IStorage` returns `QueryProcessingStage` – information about what parts of the query were already calculated inside storage.

## Parsers

A hand-written recursive descent parser parses a query. For example, `ParserSelectQuery` just recursively calls the underlying parsers for various parts of the query. Parsers create an `AST`. The `AST` is represented by nodes, which are instances of `IAST`.

Parser generators are not used for historical reasons.

## Interpreters

Interpreters are responsible for creating the query execution pipeline from an `AST`. There are simple interpreters, such as `InterpreterExistsQuery` and `InterpreterDropQuery`, or the more sophisticated `InterpreterSelectQuery`. The query execution pipeline is a combination of block input or output streams. For example, the result of interpreting the `SELECT` query is the `IBlockInputStream` to read the result set from; the result of the `INSERT` query is the `IBlockOutputStream` to write data for insertion to, and the result of interpreting the `INSERT SELECT` query is the `IBlockInputStream` that returns an empty result set on the first read, but that copies data from `SELECT` to `INSERT` at the same time.

`InterpreterSelectQuery` uses `ExpressionAnalyzer` and `ExpressionActions` machinery for query analysis and transformations. This is where most rule-based query optimizations are done. `ExpressionAnalyzer` is quite messy and should be rewritten: various query transformations and optimizations should be extracted to separate classes to allow modular transformations of query.

## Functions

There are ordinary functions and aggregate functions. For aggregate functions, see the next section.

Ordinary functions do not change the number of rows – they work as if they are processing each row independently. In fact, functions are not called for individual rows, but for `Block`'s of data to implement vectorized query execution.

There are some miscellaneous functions, like `blockSize`, `rowNumberInBlock`, and `runningAccumulate`, that exploit block processing and violate the independence of rows.

ClickHouse has strong typing, so there's no implicit type conversion. If a function does not support a specific combination of types, it throws an exception. But functions can work (be overloaded) for many different combinations of types. For example, the plus function (to implement the `+` operator) works for any combination of numeric types: `UInt8 + Float32`, `UInt16 + Int8`, and so on. Also, some variadic functions can accept any number of arguments, such as the `concat` function.

Implementing a function may be slightly inconvenient because a function explicitly dispatches supported data types and supported `IColumns`. For example, the `plus` function has code generated by instantiation of a C++ template for each combination of numeric types, and constant or non-constant left and right arguments.

It is an excellent place to implement runtime code generation to avoid template code bloat. Also, it makes it possible to add fused functions like fused multiply-add or to make multiple comparisons in one loop iteration.

Due to vectorized query execution, functions are not short-circuited. For example, if you write WHERE  $f(x)$  AND  $g(y)$ , both sides are calculated, even for rows, when  $f(x)$  is zero (except when  $f(x)$  is a zero constant expression). But if the selectivity of the  $f(x)$  condition is high, and calculation of  $f(x)$  is much cheaper than

$g(y)$ , it's better to implement multi-pass calculation. It would first calculate  $f(x)$ , then filter columns by the result, and then calculate  $g(y)$  only for smaller, filtered chunks of data.

## Aggregate Functions

Aggregate functions are stateful functions. They accumulate passed values into some state and allow you to get results from that state. They are managed with the `IAggregateFunction` interface. States can be rather simple (the state for `AggregateFunctionCount` is just a single `UInt64` value) or quite complex (the state of `AggregateFunctionUniqCombined` is a combination of a linear array, a hash table, and a `HyperLogLog` probabilistic data structure).

States are allocated in `Arena` (a memory pool) to deal with multiple states while executing a high-cardinality `GROUP BY` query. States can have a non-trivial constructor and destructor: for example, complicated aggregation states can allocate additional memory themselves. It requires some attention to creating and destroying states and properly passing their ownership and destruction order.

Aggregation states can be serialized and deserialized to pass over the network during distributed query execution or to write them on the disk where there is not enough RAM. They can even be stored in a table with the `DataTypeAggregateFunction` to allow incremental aggregation of data.

The serialized data format for aggregate function states is not versioned right now. It is ok if aggregate states are only stored temporarily. But we have the `AggregatingMergeTree` table engine for incremental aggregation, and people are already using it in production. It is the reason why backward compatibility is required when changing the serialized format for any aggregate function in the future.

## Server

The server implements several different interfaces:

- An HTTP interface for any foreign clients.
- A TCP interface for the native ClickHouse client and for cross-server communication during distributed query execution.
- An interface for transferring data for replication.

Internally, it is just a primitive multithreaded server without coroutines or fibers. Since the server is not designed to process a high rate of simple queries but to process a relatively low rate of complex queries, each of them can process a vast amount of data for analytics.

The server initializes the `Context` class with the necessary environment for query execution: the list of available databases, users and access rights, settings, clusters, the process list, the query log, and so on. Interpreters use this environment.

We maintain full backward and forward compatibility for the server TCP protocol: old clients can talk to new servers, and new clients can talk to old servers. But we do not want to maintain it eternally, and we are removing support for old versions after about one year.

### Note

For most external applications, we recommend using the HTTP interface because it is simple and easy to use. The TCP protocol is more tightly linked to internal data structures: it uses an internal format for passing blocks of data, and it uses custom framing for compressed data. We haven't released a C library for that protocol because it requires linking most of the ClickHouse codebase, which is not practical.

# Distributed Query Execution

Servers in a cluster setup are mostly independent. You can create a `Distributed` table on one or all servers in a cluster. The `Distributed` table does not store data itself – it only provides a “view” to all local tables on multiple nodes of a cluster. When you `SELECT` from a `Distributed` table, it rewrites that query, chooses remote nodes according to load balancing settings, and sends the query to them. The `Distributed` table requests remote servers to process a query just up to a stage where intermediate results from different servers can be merged. Then it receives the intermediate results and merges them. The distributed table tries to distribute as much work as possible to remote servers and does not send much intermediate data over the network.

Things become more complicated when you have subqueries in `IN` or `JOIN` clauses, and each of them uses a `Distributed` table. We have different strategies for the execution of these queries.

There is no global query plan for distributed query execution. Each node has its local query plan for its part of the job. We only have simple one-pass distributed query execution: we send queries for remote nodes and then merge the results. But this is not feasible for complicated queries with high cardinality `GROUP BYs` or with a large amount of temporary data for `JOIN`. In such cases, we need to “reshuffle” data between servers, which requires additional coordination. ClickHouse does not support that kind of query execution, and we need to work on it.

## Merge Tree

`MergeTree` is a family of storage engines that supports indexing by primary key. The primary key can be an arbitrary tuple of columns or expressions. Data in a `MergeTree` table is stored in “parts”. Each part stores data in the primary key order, so data is ordered lexicographically by the primary key tuple. All the table columns are stored in separate `column.bin` files in these parts. The files consist of compressed blocks. Each block is usually from 64 KB to 1 MB of uncompressed data, depending on the average value size. The blocks consist of column values placed contiguously one after the other. Column values are in the same order for each column (the primary key defines the order), so when you iterate by many columns, you get values for the corresponding rows.

The primary key itself is “sparse”. It does not address every single row, but only some ranges of data. A separate `primary.idx` file has the value of the primary key for each N-th row, where N is called `index_granularity` (usually, N = 8192). Also, for each column, we have `column.mrk` files with “marks”, which are offsets to each N-th row in the data file. Each mark is a pair: the offset in the file to the beginning of the compressed block, and the offset in the decompressed block to the beginning of data. Usually, compressed blocks are aligned by marks, and the offset in the decompressed block is zero. Data for `primary.idx` always resides in memory, and data for `column.mrk` files is cached.

When we are going to read something from a part in `MergeTree`, we look at `primary.idx` data and locate ranges that could contain requested data, then look at `column.mrk` data and calculate offsets for where to start reading those ranges. Because of sparseness, excess data may be read. ClickHouse is not suitable for a high load of simple point queries, because the entire range with `index_granularity` rows must be read for each key, and the entire compressed block must be decompressed for each column. We made the index sparse because we must be able to maintain trillions of rows per single server without noticeable memory consumption for the index. Also, because the primary key is sparse, it is not unique: it cannot check the existence of the key in the table at `INSERT` time. You could have many rows with the same key in a table.

When you `INSERT` a bunch of data into `MergeTree`, that bunch is sorted by primary key order and forms a new part. There are background threads that periodically select some parts and merge them into a single sorted part to keep the number of parts relatively low. That’s why it is called `MergeTree`. Of course, merging leads to “write amplification”. All parts are immutable: they are only created and deleted, but not modified. When `SELECT` is executed, it holds a snapshot of the table (a set of parts). After merging, we also keep old parts for some time to make a recovery after failure easier, so if we see that some merged part is probably broken, we can replace it with its source parts.

MergeTree is not an LSM tree because it does not contain “memtable” and “log”: inserted data is written directly to the filesystem. This makes it suitable only to INSERT data in batches, not by individual row and not very frequently – about once per second is ok, but a thousand times a second is not. We did it this way for simplicity’s sake, and because we are already inserting data in batches in our applications.

There are MergeTree engines that are doing additional work during background merges. Examples are CollapsingMergeTree and AggregatingMergeTree. This could be treated as special support for updates. Keep in mind that these are not real updates because users usually have no control over the time when background merges are executed, and data in a MergeTree table is almost always stored in more than one part, not in completely merged form.

## Replication

Replication in ClickHouse can be configured on a per-table basis. You could have some replicated and some non-replicated tables on the same server. You could also have tables replicated in different ways, such as one table with two-factor replication and another with three-factor.

Replication is implemented in the ReplicatedMergeTree storage engine. The path in ZooKeeper is specified as a parameter for the storage engine. All tables with the same path in ZooKeeper become replicas of each other: they synchronize their data and maintain consistency. Replicas can be added and removed dynamically simply by creating or dropping a table.

Replication uses an asynchronous multi-master scheme. You can insert data into any replica that has a session with ZooKeeper, and data is replicated to all other replicas asynchronously. Because ClickHouse does not support UPDATEs, replication is conflict-free. As there is no quorum acknowledgment of inserts, just-inserted data might be lost if one node fails.

Metadata for replication is stored in ZooKeeper. There is a replication log that lists what actions to do. Actions are: get part; merge parts; drop a partition, and so on. Each replica copies the replication log to its queue and then executes the actions from the queue. For example, on insertion, the “get the part” action is created in the log, and every replica downloads that part. Merges are coordinated between replicas to get byte-identical results. All parts are merged in the same way on all replicas. One of the leaders initiates a new merge first and writes “merge parts” actions to the log. Multiple replicas (or all) can be leaders at the same time. A replica can be prevented from becoming a leader using the `merge_tree` setting `replicated_can_become_leader`. The leaders are responsible for scheduling background merges.

Replication is physical: only compressed parts are transferred between nodes, not queries. Merges are processed on each replica independently in most cases to lower the network costs by avoiding network amplification. Large merged parts are sent over the network only in cases of significant replication lag.

Besides, each replica stores its state in ZooKeeper as the set of parts and its checksums. When the state on the local filesystem diverges from the reference state in ZooKeeper, the replica restores its consistency by downloading missing and broken parts from other replicas. When there is some unexpected or broken data in the local filesystem, ClickHouse does not remove it, but moves it to a separate directory and forgets it.

## Note

The ClickHouse cluster consists of independent shards, and each shard consists of replicas. The cluster is **not elastic**, so after adding a new shard, data is not rebalanced between shards automatically. Instead, the cluster load is supposed to be adjusted to be uneven. This implementation gives you more control, and it is ok for relatively small clusters, such as tens of nodes. But for clusters with hundreds of nodes that we are using in production, this approach becomes a significant drawback. We should implement a table engine that spans across the cluster with dynamically replicated regions that could be split and balanced between clusters automatically.

# Continuous Integration Checks

When you submit a pull request, some automated checks are ran for your code by the ClickHouse [continuous integration \(CI\) system](#).

This happens after a repository maintainer (someone from ClickHouse team) has screened your code and added the `can be tested` label to your pull request.

The results of the checks are listed on the GitHub pull request page as described in the [GitHub checks documentation](#).

If a check is failing, you might be required to fix it. This page gives an overview of checks you may encounter, and what you can do to fix them.

If it looks like the check failure is not related to your changes, it may be some transient failure or an infrastructure problem. Push an empty commit to the pull request to restart the CI checks:

```
git reset  
git commit --allow-empty  
git push
```

If you are not sure what to do, ask a maintainer for help.

## Merge With Master

Verifies that the PR can be merged to master. If not, it will fail with the message 'Cannot fetch mergecommit'. To fix this check, resolve the conflict as described in the [GitHub documentation](#),

or merge the master branch to your pull request branch using git.

## Docs check

Tries to build the ClickHouse documentation website. It can fail if you changed something in the documentation. Most probable reason is that some cross-link in the documentation is wrong. Go to the check report and look for `ERROR` and `WARNING` messages.

## Report Details

- [Status page example](#)
- `docs_output.txt` contains the building log. [Successful result example](#)

## Description Check

Check that the description of your pull request conforms to the template [PULL\\_REQUEST\\_TEMPLATE.md](#).

You have to specify a changelog category for your change (e.g., Bug Fix), and write a user-readable message describing the change for [CHANGELOG.md](#)

## Push To Dockerhub

Builds docker images used for build and tests, then pushes them to DockerHub.

## Marker Check

This check means that the CI system started to process the pull request. When it has 'pending' status, it means that not all checks have been started yet. After all checks have been started, it changes status to 'success'.

## Style Check

Performs some simple regex-based checks of code style, using the `utils/check-style/check-style` binary (note that it can be run locally).

If it fails, fix the style errors following the [code style guide](#).

## Report Details

- [Status page example](#)
- `output.txt` contains the check resulting errors (invalid tabulation etc), blank page means no errors.  
[Successful result example](#).

## PVS Check

Check the code with [PVS-studio](#), a static analysis tool. Look at the report to see the exact errors. Fix them if you can, if not -- ask a ClickHouse maintainer for help.

## Report Details

- [Status page example](#)
- `test_run.txt.out.log` contains the building and analyzing log file. It includes only parsing or not-found errors.
- `HTML report` contains the analysis results. For its description visit PVS's [official site](#).

## Fast Test

Normally this is the first check that is ran for a PR. It builds ClickHouse and runs most of [stateless functional tests](#), omitting some. If it fails, further checks are not started until it is fixed. Look at the report to see which tests fail, then reproduce the failure locally as described [here](#).

## Report Details

### Status page example

### Status Page Files

- `runlog.out.log` is the general log that includes all other logs.
- `test_log.txt`
- `submodule_log.txt` contains the messages about cloning and checkout needed submodules.
- `stderr.log`
- `stdout.log`
- `clickhouse-server.log`
- `clone_log.txt`
- `install_log.txt`
- `clickhouse-server.err.log`

- `build_log.txt`
- `cmake_log.txt` contains messages about the C/C++ and Linux flags check.

## Status Page Columns

- *Test name* contains the name of the test (without the path e.g. all types of tests will be stripped to the name).
- *Test status* -- one of *Skipped*, *Success*, or *Fail*.
- *Test time*, sec. -- empty on this test.

## Build Check

Builds ClickHouse in various configurations for use in further steps. You have to fix the builds that fail. Build logs often has enough information to fix the error, but you might have to reproduce the failure locally. The `cmake` options can be found in the build log, grepping for `cmake`. Use these options and follow the [general build process](#).

## Report Details

[Status page example](#).

- **Compiler:** `gcc-9` or `clang-10` (or `clang-10-xx` for other architectures e.g. `clang-10-freebsd`).
- **Build type:** `Debug` or `RelWithDebInfo` (`cmake`).
- **Sanitizer:** `none` (without sanitizers), `address` (ASan), `memory` (MSan), `undefined` (UBSan), or `thread` (TSan).
- **Bundled:** `bundled` build uses libraries from `contrib` folder, and `unbundled` build uses system libraries.
- **Splitted** `split` is a [split build](#)
- **Status:** `success` or `fail`
- **Build log:** link to the building and files copying log, useful when build failed.
- **Build time.**

- **Artifacts**: build result files (with XXX being the server version e.g. 20.8.1.4344).
  - clickhouse-client\_XXX\_all.deb
  - clickhouse-common-static-dbg\_XXX[+asan, +msan, +ubsan, +tsan]\_amd64.deb
  - clickhouse-common-staticXXX\_amd64.deb
  - clickhouse-server\_XXX\_all.deb
  - clickhouse-test\_XXX\_all.deb
  - clickhouse\_XXX\_amd64.buildinfo
  - clickhouse\_XXX\_amd64.changes
  - clickhouse: Main built binary.
  - clickhouse-odbc-bridge
- unit\_tests\_dbms: GoogleTest binary with ClickHouse unit tests.
- shared\_build.tgz: build with shared libraries.
- performance.tgz: Special package for performance tests.

## Special Build Check

Performs static analysis and code style checks using `clang-tidy`. The report is similar to the [build check](#). Fix the errors found in the build log.

## Functional Stateless Tests

Runs [stateless functional tests](#) for ClickHouse binaries built in various configurations -- release, debug, with sanitizers, etc. Look at the report to see which tests fail, then reproduce the failure locally as described [here](#). Note that you have to use the correct build configuration to reproduce -- a test might fail under AddressSanitizer but pass in Debug. Download the binary from [CI build checks page](#), or build it locally.

## Functional Stateful Tests

Runs [stateful functional tests](#). Treat them in the same way as the functional stateless tests. The difference is that they require `hits` and `visits` tables from the [Yandex.Metrica dataset](#) to run.

## Integration Tests

Runs [integration tests](#).

## Testflows Check

Runs some tests using Testflows test system. See [here](#) how to run them locally.

## Stress Test

Runs stateless functional tests concurrently from several clients to detect concurrency-related errors. If it fails:

- \* Fix all other test failures first;
- \* Look at the report to find the server logs and check them for possible causes of error.

## Split Build Smoke Test

Checks that the server build in [split build](#) configuration can start and run simple queries. If it fails:

- \* Fix other test errors first;
- \* Build the server in [split build](#development-build-md) configuration locally and check whether it can start and run `select 1`.

## Compatibility Check

Checks that `clickhouse` binary runs on distributions with old libc versions. If it fails, ask a maintainer for help.

## AST Fuzzer

Runs randomly generated queries to catch program errors. If it fails, ask a maintainer for help.

## Performance Tests

Measure changes in query performance. This is the longest check that takes just below 6 hours to run. The performance test report is described in detail [here](#).

## QA

What is a `Task (private network)` item on status pages?

It's a link to the Yandex's internal job system. Yandex employees can see the check's start time and its more verbose status.

Where the tests are run

Somewhere on Yandex internal infrastructure.

## How to Build ClickHouse on Linux

Supported platforms:

- x86\_64
- AArch64
- Power9 (experimental)

## Normal Build for Development on Ubuntu

The following tutorial is based on the Ubuntu Linux system. With appropriate changes, it should also work on any other Linux distribution.

### Install Git, CMake, Python and Ninja

```
$ sudo apt-get install git cmake python ninja-build
```

Or cmake3 instead of cmake on older systems.

## Install clang-13 (recommended)

On Ubuntu/Debian you can use the automatic installation script (check [official webpage](#))

```
sudo bash -c "$(wget -O - https://apt.llvm.org/llvm.sh)"
```

For other Linux distribution - check the availability of the [prebuild packages](#) or build clang [from sources](#).

### Use clang-13 for Builds

```
$ export CC=clang-13
$ export CXX=clang++-13
```

Gcc can also be used though it is discouraged.

## Checkout ClickHouse Sources

```
$ git clone --recursive git@github.com:ClickHouse/ClickHouse.git
```

or

```
$ git clone --recursive https://github.com/ClickHouse/ClickHouse.git
```

## Build ClickHouse

```
$ cd ClickHouse
$ mkdir build
$ cd build
$ cmake ..
$ ninja
```

To create an executable, run `ninja clickhouse`.

This will create the `programs/clickhouse` executable, which can be used with client or server arguments.

## How to Build ClickHouse on Any Linux

The build requires the following components:

- Git (is used only to checkout the sources, it's not needed for the build)
- CMake 3.10 or newer
- Ninja
- C++ compiler: clang-11 or newer
- Linker: lld
- Python (is only used inside LLVM build and it is optional)

If all the components are installed, you may build in the same way as the steps above.

Example for Ubuntu Eoan:

```
sudo apt update
sudo apt install git cmake ninja-build clang++ python
git clone --recursive https://github.com/ClickHouse/ClickHouse.git
mkdir build && cd build
cmake ..../ClickHouse
ninja
```

Example for OpenSUSE Tumbleweed:

```
sudo zypper install git cmake ninja clang-c++ python lld
git clone --recursive https://github.com/ClickHouse/ClickHouse.git
mkdir build && cd build
cmake ..../ClickHouse
ninja
```

Example for Fedora Rawhide:

```
sudo yum update
yum --nogpg install git cmake make clang-c++ python3
git clone --recursive https://github.com/ClickHouse/ClickHouse.git
mkdir build && cd build
cmake ..../ClickHouse
make -j $(nproc)
```

## How to Build ClickHouse Debian Package

### Install Git

```
$ sudo apt-get update
$ sudo apt-get install git python debhelper lsb-release fakeroot sudo debian-archive-keyring debian-keyring
```

### Checkout ClickHouse Sources

```
$ git clone --recursive --branch master https://github.com/ClickHouse/ClickHouse.git
$ cd ClickHouse
```

### Run Release Script

```
$ ./release
```

## Faster builds for development

Normally all tools of the ClickHouse bundle, such as `clickhouse-server`, `clickhouse-client` etc., are linked into a single static executable, `clickhouse`. This executable must be re-linked on every change, which might be slow. One common way to improve build time is to use the 'split' build configuration, which builds a separate binary for every tool, and further splits the code into several shared libraries. To enable this tweak, pass the following flags to `cmake`:

```
-DUSE_STATIC_LIBRARIES=0 -DSPLIT_SHARED_LIBRARIES=1 -DCCLICKHOUSE_SPLIT_BINARY=1
```

## You Don't Have to Build ClickHouse

ClickHouse is available in pre-built binaries and packages. Binaries are portable and can be run on any Linux flavour.

They are built for stable, prestable and testing releases as long as for every commit to master and for every pull request.

To find the freshest build from master, go to [commits page](#), click on the first green checkmark or red cross near commit, and click to the “Details” link right after “ClickHouse Build Check”.

## Split build configuration

Normally ClickHouse is statically linked into a single static `clickhouse` binary with minimal dependencies. This is convenient for distribution, but it means that on every change the entire binary is linked again, which is slow and may be inconvenient for development. There is an alternative configuration which creates dynamically loaded shared libraries instead, allowing faster incremental builds. To use it, add the following flags to your `cmake` invocation:

```
-DUSE_STATIC_LIBRARIES=0 -DSPLIT_SHARED_LIBRARIES=1 -DCCLICKHOUSE_SPLIT_BINARY=1
```

Note that the split build has several drawbacks: \* There is no single `clickhouse` binary, and you have to run `clickhouse-server`, `clickhouse-client`, etc. \* Risk of segfault if you run any of the programs while rebuilding the project. \* You cannot run the integration tests since they only work a single complete binary. \* You can't easily copy the binaries elsewhere. Instead of moving a single binary you'll need to copy all binaries and libraries.

## You don't have to build ClickHouse

You can install ClickHouse as follows: <https://clickhouse.com/#quick-start>

Choose Mac x86 or M1.

## How to Build ClickHouse on Mac OS X

Build should work on x86\_64 (Intel) and arm64 (Apple Silicon) based macOS 10.15 (Catalina) and higher with Homebrew's vanilla Clang.

It is always recommended to use `clang` compiler. It is possible to use XCode's `AppleClang` or `gcc` but it's strongly discouraged.

### Install Homebrew

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"  
## ...and follow the printed instructions on any additional steps required to complete the installation.
```

### Install Xcode and Command Line Tools

Install the latest [Xcode](#) from App Store.

Open it at least once to accept the end-user license agreement and automatically install the required components.

Then, make sure that the latest Command Line Tools are installed and selected in the system:

```
sudo rm -rf /Library/Developer/CommandLineTools  
sudo xcode-select --install
```

Reboot.

## Install Required Compilers, Tools, and Libraries

```
brew update  
brew install cmake ninja libtool gettext llvm gcc binutils
```

## Checkout ClickHouse Sources

```
git clone --recursive git@github.com:ClickHouse/ClickHouse.git  
## ...alternatively, you can use https://github.com/ClickHouse/ClickHouse.git as the repo URL.
```

## Build ClickHouse

To build using Homebrew's vanilla Clang compiler:

```
cd ClickHouse  
rm -rf build  
mkdir build  
cd build  
cmake -DCMAKE_C_COMPILER=$(brew --prefix llvm)/bin/clang -DCMAKE_CXX_COMPILER=$(brew --prefix  
llvm)/bin/clang++ -DCMAKE_BUILD_TYPE=RelWithDebInfo ..  
cmake --build . --config RelWithDebInfo  
cd ..
```

To build using Xcode's native AppleClang compiler (this option is strongly not recommended; use the option above):

```
cd ClickHouse  
rm -rf build  
mkdir build  
cd build  
cmake -DCMAKE_BUILD_TYPE=RelWithDebInfo ..  
cmake --build . --config RelWithDebInfo  
cd ..
```

To build using Homebrew's vanilla GCC compiler (this option is absolutely not recommended, I'm wondering why do we ever have it):

```
cd ClickHouse  
rm -rf build  
mkdir build  
cd build  
cmake -DCMAKE_C_COMPILER=$(brew --prefix gcc)/bin/gcc-11 -DCMAKE_CXX_COMPILER=$(brew --prefix  
gcc)/bin/g++-11 -DCMAKE_BUILD_TYPE=RelWithDebInfo ..  
cmake --build . --config RelWithDebInfo  
cd ..
```

## Caveats

If you intend to run clickhouse-server, make sure to increase the system's maxfiles variable.

### Note

You'll need to use sudo.

To do so, create the /Library/LaunchDaemons/limit.maxfiles.plist file with the following content:

```
<?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>
```

Give the file correct permissions:

```
sudo chown root:wheel /Library/LaunchDaemons/limit.maxfiles.plist
```

Validate that the file is correct:

```
plutil /Library/LaunchDaemons/limit.maxfiles.plist
```

Load the file (or reboot):

```
sudo launchctl load -w /Library/LaunchDaemons/limit.maxfiles.plist
```

To check if it's working, use the `ulimit -n` or `launchctl limit maxfiles` commands.

## Run ClickHouse server:

```
cd ClickHouse
./build/programs/clickhouse-server --config-file ./programs/server/config.xml
```

# How to Build ClickHouse on Linux for Mac OS X

This is for the case when you have Linux machine and want to use it to build clickhouse binary that will run on OS X. This is intended for continuous integration checks that run on Linux servers. If you want to build ClickHouse directly on Mac OS X, then proceed with [another instruction](#).

The cross-build for Mac OS X is based on the [Build instructions](#), follow them first.

## Install Clang-8

Follow the instructions from <https://apt.llvm.org/> for your Ubuntu or Debian setup.

For example the commands for Bionic are like:

```
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
```

# Install Cross-Compilation Toolset

Let's remember the path where we install `cctools` as  `${CCTOOLS}`

```
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
```

Also, we need to download macOS X SDK into the working tree.

```
cd ClickHouse
wget 'https://github.com/phracker/MacOSX-SDKs/releases/download/10.15/MacOSX10.15.sdk.tar.xz'
mkdir -p build-darwin/cmake/toolchain/darwin-x86_64
tar xJf MacOSX10.15.sdk.tar.xz -C build-darwin/cmake/toolchain/darwin-x86_64 --strip-components=1
```

## Build ClickHouse

```
cd ClickHouse
mkdir build-osx
CC=clang-8 CXX=clang++-8 cmake . -Bbuild-osx -DCMAKE_TOOLCHAIN_FILE=cmake/darwin/toolchain-x86_64.cmake \
-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
ninja -C build-osx
```

The resulting binary will have a Mach-O executable format and can't be run on Linux.

## How to Build ClickHouse on Linux for AARCH64 (ARM64) Architecture

This is for the case when you have Linux machine and want to use it to build `clickhouse` binary that will run on another Linux machine with AARCH64 CPU architecture. This is intended for continuous integration checks that run on Linux servers.

The cross-build for AARCH64 is based on the [Build instructions](#), follow them first.

## Install Clang-13

Follow the instructions from <https://apt.llvm.org/> for your Ubuntu or Debian setup or do

```
sudo bash -c "$(wget -O - https://apt.llvm.org/llvm.sh)"
```

## Install Cross-Compilation Toolset

```
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 xf gcc-arm-8.3-2019.03-x86_64-aarch64-linux-gnu.tar.xz -C build-aarch64/cmake/toolchain/linux-aarch64 --strip-components=1
```

## Build ClickHouse

```
cd ClickHouse
mkdir build-arm64
CC=clang-13 CXX=clang++-13 cmake . -Bbuild-arm64 -DCMAKE_TOOLCHAIN_FILE=cmake/linux/toolchain-aarch64.cmake
ninja -C build-arm64
```

The resulting binary will run only on Linux with the AARCH64 CPU architecture.

## How to Write C++ Code

### General Recommendations

- 1.** The following are recommendations, not requirements.
- 2.** If you are editing code, it makes sense to follow the formatting of the existing code.
- 3.** Code style is needed for consistency. Consistency makes it easier to read the code, and it also makes it easier to search the code.
- 4.** Many of the rules do not have logical reasons; they are dictated by established practices.

### Formatting

- 1.** Most of the formatting will be done automatically by `clang-format`.
- 2.** Indents are 4 spaces. Configure your development environment so that a tab adds four spaces.
- 3.** Opening and closing curly brackets must be on a separate line.

```
inline void readBoolText(bool & x, ReadBuffer & buf)
{
    char tmp = '0';
    readChar(tmp, buf);
    x = tmp != '0';
}
```

- 4.** If the entire function body is a single `statement`, it can be placed on a single line. Place spaces around curly braces (besides the space at the end of the line).

```
inline size_t mask() const { return buf_size() - 1; }
inline size_t place(HashValue x) const { return x & mask(); }
```

- 5.** For functions. Don't put spaces around brackets.

```
void reinsert(const Value & x)
```

```
memcpy(&buf[place_value], &x, sizeof(x));
```

- 6.** In `if`, `for`, `while` and other expressions, a space is inserted in front of the opening bracket (as opposed to function calls).

```
for (size_t i = 0; i < rows; i += storage.index_granularity)
```

- 7.** Add spaces around binary operators (`+`, `-`, `*`, `/`, `%`, ...) and the ternary operator `?:`.

```
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');
```

- 8.** If a line feed is entered, put the operator on a new line and increase the indent before it.

```
if (elapsed_ns)  
    message << "("  
        << rows_read_on_server * 1000000000 / elapsed_ns << " rows/s., "  
        << bytes_read_on_server * 1000.0 / elapsed_ns << " MB/s.) ";
```

- 9.** You can use spaces for alignment within a line, if desired.

```
dst.ClickLogID      = click.LogID;  
dst.ClickEventID    = click.EventID;  
dst.ClickGoodEvent  = click.GoodEvent;
```

- 10.** Don't use spaces around the operators `,`, `->`.

If necessary, the operator can be wrapped to the next line. In this case, the offset in front of it is increased.

- 11.** Do not use a space to separate unary operators (`-`, `++`, `*`, `&`, ...) from the argument.

- 12.** Put a space after a comma, but not before it. The same rule goes for a semicolon inside a `for` expression.

- 13.** Do not use spaces to separate the `[]` operator.

- 14.** In a template `<...>` expression, use a space between template and `<`; no spaces after `<` or before `>`.

```
template <typename TKey, typename TValue>  
struct AggregatedStatElement  
{}
```

- 15.** In classes and structures, write `public`, `private`, and `protected` on the same level as `class/struct`, and indent the rest of the code.

```
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.** If the same `namespace` is used for the entire file, and there isn't anything else significant, an offset is not necessary inside `namespace`.

**17.** If the block for an `if`, `for`, `while`, or other expression consists of a single `statement`, the curly brackets are optional. Place the `statement` on a separate line, instead. This rule is also valid for nested `if`, `for`, `while`, ...

But if the inner `statement` contains curly brackets or `else`, the external block should be written in curly brackets.

```
/// Finish write.  
for (auto & stream : streams)  
    stream.second->finalize();
```

**18.** There shouldn't be any spaces at the ends of lines.

**19.** Source files are UTF-8 encoded.

**20.** Non-ASCII characters can be used in string literals.

```
<< ", " << (timer.elapsed() / chunks_stats.hits) << " µsec(hit.>";
```

**21.** Do not write multiple expressions in a single line.

**22.** Group sections of code inside functions and separate them with no more than one empty line.

**23.** Separate functions, classes, and so on with one or two empty lines.

**24.** A `const` (related to a value) must be written before the type name.

```
//correct  
const char * pos  
const std::string & s  
//incorrect  
char const * pos
```

**25.** When declaring a pointer or reference, the `*` and `&` symbols should be separated by spaces on both sides.

```
//correct  
const char * pos  
//incorrect  
const char* pos  
const char *pos
```

**26.** When using template types, alias them with the `using` keyword (except in the simplest cases).

In other words, the template parameters are specified only in `using` and aren't repeated in the code.

`using` can be declared locally, such as inside a function.

```
//correct  
using FileStreams = std::map<std::string, std::shared_ptr<Stream>>;  
FileStreams streams;  
//incorrect  
std::map<std::string, std::shared_ptr<Stream>> streams;
```

**27.** Do not declare several variables of different types in one statement.

```
//incorrect  
int x, *y;
```

## **28.** Do not use C-style casts.

```
//incorrect  
std::cerr << (int)c << std::endl;  
//correct  
std::cerr << static_cast<int>(c) << std::endl;
```

## **29.** In classes and structs, group members and functions separately inside each visibility scope.

## **30.** For small classes and structs, it is not necessary to separate the method declaration from the implementation.

The same is true for small methods in any classes or structs.

For templated classes and structs, do not separate the method declarations from the implementation (because otherwise they must be defined in the same translation unit).

## **31.** You can wrap lines at 140 characters, instead of 80.

## **32.** Always use the prefix increment/decrement operators if postfix is not required.

```
for (Names::const_iterator it = column_names.begin(); it != column_names.end(); ++it)
```

# Comments

## **1.** Be sure to add comments for all non-trivial parts of code.

This is very important. Writing the comment might help you realize that the code isn't necessary, or that it is designed wrong.

```
/** 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.** Comments can be as detailed as necessary.

## **3.** Place comments before the code they describe. In rare cases, comments can come after the code, on the same line.

```
/** 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.** Comments should be written in English only.

## **5.** If you are writing a library, include detailed comments explaining it in the main header file.

## **6.** Do not add comments that do not provide additional information. In particular, do not leave empty comments like this:

```
/*
 * 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:
 */
```

The example is borrowed from the resource <http://home.tamk.fi/~jaalto/course/coding-style/doc/unmaintainable-code/>.

**7.** Do not write garbage comments (author, creation date ..) at the beginning of each file.

**8.** Single-line comments begin with three slashes: /// and multi-line comments begin with /\*\*. These comments are considered “documentation”.

Note: You can use Doxygen to generate documentation from these comments. But Doxygen is not generally used because it is more convenient to navigate the code in the IDE.

**9.** Multi-line comments must not have empty lines at the beginning and end (except the line that closes a multi-line comment).

**10.** For commenting out code, use basic comments, not “documenting” comments.

**11.** Delete the commented out parts of the code before committing.

**12.** Do not use profanity in comments or code.

**13.** Do not use uppercase letters. Do not use excessive punctuation.

```
/// WHAT THE FAIL???
```

**14.** Do not use comments to make delimiters.

```
//
```

**15.** Do not start discussions in comments.

```
/// Why did you do this stuff?
```

**16.** There's no need to write a comment at the end of a block describing what it was about.

```
/// for
```

## Names

**1.** Use lowercase letters with underscores in the names of variables and class members.

```
size_t max_block_size;
```

**2.** For the names of functions (methods), use camelCase beginning with a lowercase letter.

```
std::string getName() const override { return "Memory"; }
```

**3.** For the names of classes (structs), use CamelCase beginning with an uppercase letter. Prefixes other than I are not used for interfaces.

```
class StorageMemory : public IStorage
```

**4.** `using` are named the same way as classes, or with `_t` on the end.

**5.** Names of template type arguments: in simple cases, use `T`; `T`, `U`; `T1`, `T2`.

For more complex cases, either follow the rules for class names, or add the prefix `T`.

```
template <typename TKey, typename TValue>
struct AggregatedStatElement
```

**6.** Names of template constant arguments: either follow the rules for variable names, or use `N` in simple cases.

```
template <bool without_www>
struct ExtractDomain
```

**7.** For abstract classes (interfaces) you can add the `I` prefix.

```
class IBlockInputStream
```

**8.** If you use a variable locally, you can use the short name.

In all other cases, use a name that describes the meaning.

```
bool info_successfully_loaded = false;
```

**9.** Names of `defines` and global constants use ALL\_CAPS with underscores.

```
##define MAX_SRC_TABLE_NAMES_TO_STORE 1000
```

**10.** File names should use the same style as their contents.

If a file contains a single class, name the file the same way as the class (CamelCase).

If the file contains a single function, name the file the same way as the function (camelCase).

**11.** If the name contains an abbreviation, then:

- For variable names, the abbreviation should use lowercase letters `mysql_connection` (not `mySQL_connection`).
- For names of classes and functions, keep the uppercase letters in the abbreviation `MySQLConnection` (not `MySqlConnection`).

**12.** Constructor arguments that are used just to initialize the class members should be named the same way as the class members, but with an underscore at the end.

```
FileQueueProcessor(  
    const std::string & path_,  
    const std::string & prefix_,  
    std::shared_ptr<FileHandler> handler_ )  
: path(path_),  
prefix(prefix_),  
handler(handler_),  
log(&Logger::get("FileQueueProcessor"))  
{  
}
```

The underscore suffix can be omitted if the argument is not used in the constructor body.

**13.** There is no difference in the names of local variables and class members (no prefixes required).

```
timer (not m_timer)
```

**14.** For the constants in an enum, use CamelCase with a capital letter. ALL\_CAPS is also acceptable. If the enum is non-local, use an enum class.

```
enum class CompressionMethod  
{  
    QuickLZ = 0,  
    LZ4      = 1,  
};
```

**15.** All names must be in English. Transliteration of Russian words is not allowed.

```
not Stroka
```

**16.** Abbreviations are acceptable if they are well known (when you can easily find the meaning of the abbreviation in Wikipedia or in a search engine).

```
`AST`, `SQL`.
```

```
Not `NVDH` (some random letters)
```

Incomplete words are acceptable if the shortened version is common use.

You can also use an abbreviation if the full name is included next to it in the comments.

**17.** File names with C++ source code must have the .cpp extension. Header files must have the .h extension.

## How to Write Code

### 1. Memory management.

Manual memory deallocation (`delete`) can only be used in library code.

In library code, the `delete` operator can only be used in destructors.

In application code, memory must be freed by the object that owns it.

Examples:

- The easiest way is to place an object on the stack, or make it a member of another class.
- For a large number of small objects, use containers.
- For automatic deallocation of a small number of objects that reside in the heap, use `shared_ptr/unique_ptr`.

## 2. Resource management.

Use RAII and see above.

## 3. Error handling.

Use exceptions. In most cases, you only need to throw an exception, and do not need to catch it (because of RAII).

In offline data processing applications, it's often acceptable to not catch exceptions.

In servers that handle user requests, it's usually enough to catch exceptions at the top level of the connection handler.

In thread functions, you should catch and keep all exceptions to rethrow them in the main thread after `join`.

```
/// 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();
```

Never hide exceptions without handling. Never just blindly put all exceptions to log.

```
//Not correct
catch (...) {}
```

If you need to ignore some exceptions, do so only for specific ones and rethrow the rest.

```
catch (const DB::Exception & e)
{
    if (e.code() == ErrorCode::UNKNOWN_AGGREGATE_FUNCTION)
        return nullptr;
    else
        throw;
}
```

When using functions with response codes or `errno`, always check the result and throw an exception in case of error.

```
if (0 != close(fd))
    throwError("Cannot close file " + file_name, ErrorCode::CANNOT_CLOSE_FILE);
```

Do not use assert.

## 4. Exception types.

There is no need to use complex exception hierarchy in application code. The exception text should be understandable to a system administrator.

## 5. Throwing exceptions from destructors.

This is not recommended, but it is allowed.

Use the following options:

- Create a function (`done()` or `finalize()`) that will do all the work in advance that might lead to an exception.  
If that function was called, there should be no exceptions in the destructor later.
- Tasks that are too complex (such as sending messages over the network) can be put in separate method that the class user will have to call before destruction.
- If there is an exception in the destructor, it's better to log it than to hide it (if the logger is available).
- In simple applications, it is acceptable to rely on `std::terminate` (for cases of `noexcept` by default in C++11) to handle exceptions.

## 6. Anonymous code blocks.

You can create a separate code block inside a single function in order to make certain variables local, so that the destructors are called when exiting the block.

```
Block block = data.in->read();

{
    std::lock_guard<std::mutex> lock(mutex);
    data.ready = true;
    data.block = block;
}

ready_any.set();
```

## 7. Multithreading.

In offline data processing programs:

- Try to get the best possible performance on a single CPU core. You can then parallelize your code if necessary.

In server applications:

- Use the thread pool to process requests. At this point, we haven't had any tasks that required userspace context switching.

Fork is not used for parallelization.

## 8. Syncing threads.

Often it is possible to make different threads use different memory cells (even better: different cache lines,) and to not use any thread synchronization (except `joinAll`).

If synchronization is required, in most cases, it is sufficient to use mutex under `lock_guard`.

In other cases use system synchronization primitives. Do not use busy wait.

Atomic operations should be used only in the simplest cases.

Do not try to implement lock-free data structures unless it is your primary area of expertise.

## 9. Pointers vs references.

In most cases, prefer references.

## **10. const.**

Use constant references, pointers to constants, `const_iterator`, and `const` methods.

Consider `const` to be default and use `non-const` only when necessary.

When passing variables by value, using `const` usually does not make sense.

## **11. unsigned.**

Use `unsigned` if necessary.

## **12. Numeric types.**

Use the types `UInt8`, `UInt16`, `UInt32`, `UInt64`, `Int8`, `Int16`, `Int32`, and `Int64`, as well as `size_t`, `ssize_t`, and `ptrdiff_t`.

Don't use these types for numbers: `signed/unsigned long`, `long long`, `short`, `signed/unsigned char`, `char`.

## **13. Passing arguments.**

Pass complex values by reference (including `std::string`).

If a function captures ownership of an object created in the heap, make the argument type `shared_ptr` or `unique_ptr`.

## **14. Return values.**

In most cases, just use `return`. Do not write `return std::move(res)`.

If the function allocates an object on heap and returns it, use `shared_ptr` or `unique_ptr`.

In rare cases you might need to return the value via an argument. In this case, the argument should be a reference.

```
using AggregateFunctionPtr = std::shared_ptr<IAggregateFunction>;  
/* Allows creating an aggregate function by its name.  
 */  
class AggregateFunctionFactory  
{  
public:  
    AggregateFunctionFactory();  
    AggregateFunctionPtr get(const String & name, const DataTypes & argument_types) const;
```

## **15. namespace.**

There is no need to use a separate namespace for application code.

Small libraries do not need this, either.

For medium to large libraries, put everything in a `namespace`.

In the library's `.h` file, you can use `namespace detail` to hide implementation details not needed for the application code.

In a `.cpp` file, you can use a static or anonymous namespace to hide symbols.

Also, a `namespace` can be used for an `enum` to prevent the corresponding names from falling into an external `namespace` (but it's better to use an `enum class`).

## **16. Deferred initialization.**

If arguments are required for initialization, then you normally shouldn't write a default constructor.

If later you'll need to delay initialization, you can add a default constructor that will create an invalid object. Or, for a small number of objects, you can use `shared_ptr`/`unique_ptr`.

```
Loader(DB::Connection * connection_, const std::string & query, size_t max_block_size_);  
/// For deferred initialization  
Loader() {}
```

## 17. Virtual functions.

If the class is not intended for polymorphic use, you do not need to make functions virtual. This also applies to the destructor.

## 18. Encodings.

Use UTF-8 everywhere. Use `std::string` and `char *`. Do not use `std::wstring` and `wchar_t`.

## 19. Logging.

See the examples everywhere in the code.

Before committing, delete all meaningless and debug logging, and any other types of debug output.

Logging in cycles should be avoided, even on the Trace level.

Logs must be readable at any logging level.

Logging should only be used in application code, for the most part.

Log messages must be written in English.

The log should preferably be understandable for the system administrator.

Do not use profanity in the log.

Use UTF-8 encoding in the log. In rare cases you can use non-ASCII characters in the log.

## 20. Input-output.

Don't use `iostreams` in internal cycles that are critical for application performance (and never use `stringstream`).

Use the DB/IO library instead.

## 21. Date and time.

See the `DateLUT` library.

## 22. include.

Always use `#pragma once` instead of include guards.

## 23. using.

`using namespace` is not used. You can use `using` with something specific. But make it local inside a class or function.

## 24. Do not use trailing return type for functions unless necessary.

```
auto f() -> void
```

## 25. Declaration and initialization of variables.

```
//right way
std::string s = "Hello";
std::string s{"Hello"};

//wrong way
auto s = std::string{"Hello"};
```

**26.** For virtual functions, write `virtual` in the base class, but write `override` instead of `virtual` in descendent classes.

## Unused Features of C++

- 1.** Virtual inheritance is not used.
- 2.** Exception specifiers from C++03 are not used.

## Platform

- 1.** We write code for a specific platform.

But other things being equal, cross-platform or portable code is preferred.

- 2.** Language: C++20 (see the list of available [C++20 features](#)).
- 3.** Compiler: clang. At this time (April 2021), the code is compiled using clang version 11. (It can also be compiled using `gcc` version 10, but it's untested and not suitable for production usage).

The standard library is used (`libc++`).

- 4.** OS: Linux Ubuntu, not older than Precise.

- 5.** Code is written for x86\_64 CPU architecture.

The CPU instruction set is the minimum supported set among our servers. Currently, it is SSE 4.2.

- 6.** Use `-Wall -Wextra -Werror` compilation flags. Also `-Weverything` is used with few exceptions.
- 7.** Use static linking with all libraries except those that are difficult to connect to statically (see the output of the `ldd` command).
- 8.** Code is developed and debugged with release settings.

## Tools

- 1.** KDevelop is a good IDE.
- 2.** For debugging, use `gdb`, `valgrind (memcheck)`, `strace`, `-fsanitize=...`, or `tcmalloc_minimal_debug`.
- 3.** For profiling, use `Linux Perf`, `valgrind (callgrind)`, or `strace -cf`.
- 4.** Sources are in Git.
- 5.** Assembly uses CMake.
- 6.** Programs are released using `deb` packages.
- 7.** Commits to master must not break the build.

Though only selected revisions are considered workable.

- 8.** Make commits as often as possible, even if the code is only partially ready.

Use branches for this purpose.

If your code in the `master` branch is not buildable yet, exclude it from the build before the `push`. You'll need to finish it or remove it within a few days.

**9.** For non-trivial changes, use branches and publish them on the server.

**10.** Unused code is removed from the repository.

## Libraries

**1.** The C++20 standard library is used (experimental extensions are allowed), as well as `boost` and `Poco` frameworks.

**2.** It is not allowed to use libraries from OS packages. It is also not allowed to use pre-installed libraries. All libraries should be placed in form of source code in `contrib` directory and built with ClickHouse. See [Guidelines for adding new third-party libraries](#) for details.

**3.** Preference is always given to libraries that are already in use.

## General Recommendations

**1.** Write as little code as possible.

**2.** Try the simplest solution.

**3.** Don't write code until you know how it's going to work and how the inner loop will function.

**4.** In the simplest cases, use `using` instead of classes or structs.

**5.** If possible, do not write copy constructors, assignment operators, destructors (other than a virtual one, if the class contains at least one virtual function), move constructors or move assignment operators. In other words, the compiler-generated functions must work correctly. You can use `default`.

**6.** Code simplification is encouraged. Reduce the size of your code where possible.

## Additional Recommendations

**1.** Explicitly specifying `std::` for types from `stddef.h`

is not recommended. In other words, we recommend writing `size_t` instead `std::size_t`, because it's shorter.

It is acceptable to add `std::`:

**2.** Explicitly specifying `std::` for functions from the standard C library

is not recommended. In other words, write `memcpy` instead of `std::memcpy`.

The reason is that there are similar non-standard functions, such as `memmem`. We do use these functions on occasion. These functions do not exist in namespace `std`.

If you write `std::memcpy` instead of `memcpy` everywhere, then `memmem` without `std::` will look strange.

Nevertheless, you can still use `std::` if you prefer it.

**3.** Using functions from C when the same ones are available in the standard C++ library.

This is acceptable if it is more efficient.

For example, use `memcpy` instead of `std::copy` for copying large chunks of memory.

**4.** Multiline function arguments.

Any of the following wrapping styles are allowed:

```
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)
```

## ClickHouse Testing

### Functional Tests

Functional tests are the most simple and convenient to use. Most of ClickHouse features can be tested with functional tests and they are mandatory to use for every change in ClickHouse code that can be tested that way.

Each functional test sends one or multiple queries to the running ClickHouse server and compares the result with reference.

Tests are located in `queries` directory. There are two subdirectories: `stateless` and `stateful`. Stateless tests run queries without any preloaded test data - they often create small synthetic datasets on the fly, within the test itself. Stateful tests require preloaded test data from Yandex.Metrica and it is available to general public.

Each test can be one of two types: `.sql` and `.sh`. `.sql` test is the simple SQL script that is piped to `clickhouse-client --multiquery --testmode`. `.sh` test is a script that is run by itself. SQL tests are generally preferable to `.sh` tests. You should use `.sh` tests only when you have to test some feature that cannot be exercised from pure SQL, such as piping some input data into `clickhouse-client` or testing `clickhouse-local`.

### Running a Test Locally

Start the ClickHouse server locally, listening on the default port (9000). To run, for example, the test `01428_hash_set_nan_key`, change to the repository folder and run the following command:

```
PATH=$PATH:<path to clickhouse-client> tests/clickhouse-test 01428_hash_set_nan_key
```

For more options, see `tests/clickhouse-test --help`. You can simply run all tests or run subset of tests filtered by substring in test name: `./clickhouse-test substring`. There are also options to run tests in parallel or in randomized order.

## Adding a New Test

To add new test, create a `.sql` or `.sh` file in `queries/0_stateless` directory, check it manually and then generate `.reference` file in the following way: `clickhouse-client -n --testmode < 00000_test.sql > 00000_test.reference` or `./00000_test.sh > ./00000_test.reference`.

Tests should use (create, drop, etc) only tables in `test` database that is assumed to be created beforehand; also tests can use temporary tables.

## Choosing the Test Name

The name of the test starts with a five-digit prefix followed by a descriptive name, such as `00422_hash_function_constexpr.sql`. To choose the prefix, find the largest prefix already present in the directory, and increment it by one. In the meantime, some other tests might be added with the same numeric prefix, but this is OK and does not lead to any problems, you don't have to change it later.

Some tests are marked with `zookeeper`, `shard` or `long` in their names. `zookeeper` is for tests that are using ZooKeeper. `shard` is for tests that require server to listen `127.0.0.*`; `distributed` or `global` have the same meaning. `long` is for tests that run slightly longer than one second. You can disable these groups of tests using `--no-zookeeper`, `--no-shard` and `--no-long` options, respectively. Make sure to add a proper prefix to your test name if it needs ZooKeeper or distributed queries.

## Checking for an Error that Must Occur

Sometimes you want to test that a server error occurs for an incorrect query. We support special annotations for this in SQL tests, in the following form:

```
select x; -- { serverError 49 }
```

This test ensures that the server returns an error with code 49 about unknown column `x`. If there is no error, or the error is different, the test will fail. If you want to ensure that an error occurs on the client side, use `clientError` annotation instead.

Do not check for a particular wording of error message, it may change in the future, and the test will needlessly break. Check only the error code. If the existing error code is not precise enough for your needs, consider adding a new one.

## Testing a Distributed Query

If you want to use distributed queries in functional tests, you can leverage `remote table` function with `127.0.0.{1..2}` addresses for the server to query itself; or you can use predefined test clusters in server configuration file like `test_shard_localhost`. Remember to add the words `shard` or `distributed` to the test name, so that it is run in CI in correct configurations, where the server is configured to support distributed queries.

## Known Bugs

If we know some bugs that can be easily reproduced by functional tests, we place prepared functional tests in `tests/queries/bugs` directory. These tests will be moved to `tests/queries/0_stateless` when bugs are fixed.

## Integration Tests

Integration tests allow testing ClickHouse in clustered configuration and ClickHouse interaction with other servers like MySQL, Postgres, MongoDB. They are useful to emulate network splits, packet drops, etc. These tests are run under Docker and create multiple containers with various software.

See `tests/integration/README.md` on how to run these tests.

Note that integration of ClickHouse with third-party drivers is not tested. Also, we currently do not have integration tests with our JDBC and ODBC drivers.

## Unit Tests

Unit tests are useful when you want to test not the ClickHouse as a whole, but a single isolated library or class. You can enable or disable build of tests with `ENABLE_TESTS` CMake option. Unit tests (and other test programs) are located in `tests` subdirectories across the code. To run unit tests, type `ninja test`. Some tests use `gtest`, but some are just programs that return non-zero exit code on test failure.

It's not necessary to have unit tests if the code is already covered by functional tests (and functional tests are usually much more simple to use).

You can run individual `gtest` checks by calling the executable directly, for example:

```
$ ./src/unit_tests_dbms --gtest_filter=LocalAddress*
```

## Performance Tests

Performance tests allow to measure and compare performance of some isolated part of ClickHouse on synthetic queries. Tests are located at `tests/performance`. Each test is represented by `.xml` file with description of test case. Tests are run with `docker/tests/performance-comparison` tool . See the `readme` file for invocation.

Each test run one or multiple queries (possibly with combinations of parameters) in a loop. Some tests can contain preconditions on preloaded test dataset.

If you want to improve performance of ClickHouse in some scenario, and if improvements can be observed on simple queries, it is highly recommended to write a performance test. It always makes sense to use `perf top` or other perf tools during your tests.

## Test Tools and Scripts

Some programs in `tests` directory are not prepared tests, but are test tools. For example, for `Lexer` there is a tool `src/Parsers/tests/lexer` that just do tokenization of `stdin` and writes colorized result to `stdout`. You can use these kind of tools as a code examples and for exploration and manual testing.

## Miscellaneous Tests

There are tests for machine learned models in `tests/external_models`. These tests are not updated and must be transferred to integration tests.

There is separate test for quorum inserts. This test run ClickHouse cluster on separate servers and emulate various failure cases: network split, packet drop (between ClickHouse nodes, between ClickHouse and ZooKeeper, between ClickHouse server and client, etc.), `kill -9`, `kill -STOP` and `kill -CONT` , like `Jepsen`. Then the test checks that all acknowledged inserts was written and all rejected inserts was not.

Quorum test was written by separate team before ClickHouse was open-sourced. This team no longer work with ClickHouse. Test was accidentally written in Java. For these reasons, quorum test must be rewritten and moved to integration tests.

# Manual Testing

When you develop a new feature, it is reasonable to also test it manually. You can do it with the following steps:

Build ClickHouse. Run ClickHouse from the terminal: change directory to `programs/clickhouse-server` and run it with `./clickhouse-server`. It will use configuration (`config.xml`, `users.xml` and files within `config.d` and `users.d` directories) from the current directory by default. To connect to ClickHouse server, run `programs/clickhouse-client/clickhouse-client`.

Note that all `clickhouse` tools (server, client, etc) are just symlinks to a single binary named `clickhouse`. You can find this binary at `programs/clickhouse`. All tools can also be invoked as `clickhouse` tool instead of `clickhouse`-`tool`.

Alternatively you can install ClickHouse package: either stable release from Yandex repository or you can build package for yourself with `./release` in ClickHouse sources root. Then start the server with sudo service `clickhouse-server start` (or stop to stop the server). Look for logs at `/etc/clickhouse-server/clickhouse-server.log`.

When ClickHouse is already installed on your system, you can build a new `clickhouse` binary and replace the existing binary:

```
$ sudo service clickhouse-server stop  
$ sudo cp ./clickhouse /usr/bin/  
$ sudo service clickhouse-server start
```

Also you can stop system `clickhouse-server` and run your own with the same configuration but with logging to terminal:

```
$ sudo service clickhouse-server stop  
$ sudo -u clickhouse /usr/bin/clickhouse server --config-file /etc/clickhouse-server/config.xml
```

Example with `gdb`:

```
$ sudo -u clickhouse gdb --args /usr/bin/clickhouse server --config-file /etc/clickhouse-server/config.xml
```

If the system `clickhouse-server` is already running and you do not want to stop it, you can change port numbers in your `config.xml` (or override them in a file in `config.d` directory), provide appropriate data path, and run it.

`clickhouse` binary has almost no dependencies and works across wide range of Linux distributions. To quick and dirty test your changes on a server, you can simply `scp` your fresh built `clickhouse` binary to your server and then run it as in examples above.

## Testing Environment

Before publishing release as stable we deploy it on testing environment. Testing environment is a cluster that process 1/39 part of [Yandex.Metrica](#) data. We share our testing environment with Yandex.Metrica team. ClickHouse is upgraded without downtime on top of existing data. We look at first that data is processed successfully without lagging from realtime, the replication continue to work and there is no issues visible to Yandex.Metrica team. First check can be done in the following way:

```
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;
```

In some cases we also deploy to testing environment of our friend teams in Yandex: Market, Cloud, etc. Also we have some hardware servers that are used for development purposes.

## Load Testing

After deploying to testing environment we run load testing with queries from production cluster. This is done manually.

Make sure you have enabled `query_log` on your production cluster.

Collect query log for a day or more:

```
$ 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
```

This is a way complicated example. `type = 2` will filter queries that are executed successfully. `query LIKE '%ym:%'` is to select relevant queries from Yandex.Metrica. `is_initial_query` is to select only queries that are initiated by client, not by ClickHouse itself (as parts of distributed query processing).

scp this log to your testing cluster and run it as following:

```
$ clickhouse benchmark --concurrency 16 < queries.tsv
```

(probably you also want to specify a `--user`)

Then leave it for a night or weekend and go take a rest.

You should check that `clickhouse-server` does not crash, memory footprint is bounded and performance not degrading over time.

Precise query execution timings are not recorded and not compared due to high variability of queries and environment.

## Build Tests

Build tests allow to check that build is not broken on various alternative configurations and on some foreign systems. These tests are automated as well.

Examples:

- cross-compile for Darwin x86\_64 (Mac OS X)
- cross-compile for FreeBSD x86\_64
- cross-compile for Linux AArch64
- build on Ubuntu with libraries from system packages (discouraged)
- build with shared linking of libraries (discouraged)

For example, build with system packages is bad practice, because we cannot guarantee what exact version of packages a system will have. But this is really needed by Debian maintainers. For this reason we at least have to support this variant of build. Another example: shared linking is a common source of trouble, but it is needed for some enthusiasts.

Though we cannot run all tests on all variant of builds, we want to check at least that various build variants are not broken. For this purpose we use build tests.

We also test that there are no translation units that are too long to compile or require too much RAM.

We also test that there are no too large stack frames.

## Testing for Protocol Compatibility

When we extend ClickHouse network protocol, we test manually that old clickhouse-client works with new clickhouse-server and new clickhouse-client works with old clickhouse-server (simply by running binaries from corresponding packages).

We also test some cases automatically with integrational tests:

- if data written by old version of ClickHouse can be successfully read by the new version;
- do distributed queries work in a cluster with different ClickHouse versions.

## Help from the Compiler

Main ClickHouse code (that is located in `dbms` directory) is built with `-Wall -Wextra -Werror` and with some additional enabled warnings. Although these options are not enabled for third-party libraries.

Clang has even more useful warnings - you can look for them with `-Weverything` and pick something to default build.

For production builds, clang is used, but we also test make gcc builds. For development, clang is usually more convenient to use. You can build on your own machine with debug mode (to save battery of your laptop), but please note that compiler is able to generate more warnings with `-O3` due to better control flow and inter-procedure analysis. When building with clang in debug mode, debug version of `libc++` is used that allows to catch more errors at runtime.

## Sanitizers

### Address sanitizer

We run functional, integration, stress and unit tests under ASan on per-commit basis.

### Thread sanitizer

We run functional, integration, stress and unit tests under TSan on per-commit basis.

### Memory sanitizer

We run functional, integration, stress and unit tests under MSan on per-commit basis.

### Undefined behaviour sanitizer

We run functional, integration, stress and unit tests under UBSan on per-commit basis. The code of some third-party libraries is not sanitized for UB.

### Valgrind (Memcheck)

We used to run functional tests under Valgrind overnight, but don't do it anymore. It takes multiple hours. Currently there is one known false positive in `re2` library, see [this article](#).

## Fuzzing

ClickHouse fuzzing is implemented both using `libFuzzer` and random SQL queries.

All the fuzz testing should be performed with sanitizers (Address and Undefined).

LibFuzzer is used for isolated fuzz testing of library code. Fuzzers are implemented as part of test code and have `_fuzzer` name postfixes.

Fuzzer example can be found at `src/Parsers/tests/lexer_fuzzer.cpp`. LibFuzzer-specific configs, dictionaries and corpus are stored at `tests/fuzz`.

We encourage you to write fuzz tests for every functionality that handles user input.

Fuzzers are not built by default. To build fuzzers both `-DENABLE_FUZZING=1` and `-DENABLE_TESTS=1` options should be set.

We recommend to disable Jemalloc while building fuzzers. Configuration used to integrate ClickHouse fuzzing to

Google OSS-Fuzz can be found at [docker/fuzz](#).

We also use simple fuzz test to generate random SQL queries and to check that the server does not die executing them.

You can find it in `00746_sql_fuzzy.pl`. This test should be run continuously (overnight and longer).

We also use sophisticated AST-based query fuzzer that is able to find huge amount of corner cases. It does random permutations and substitutions in queries AST. It remembers AST nodes from previous tests to use them for fuzzing of subsequent tests while processing them in random order. You can learn more about this fuzzer in [this blog article](#).

## Stress test

Stress tests are another case of fuzzing. It runs all functional tests in parallel in random order with a single server. Results of the tests are not checked.

It is checked that:

- server does not crash, no debug or sanitizer traps are triggered;
- there are no deadlocks;
- the database structure is consistent;
- server can successfully stop after the test and start again without exceptions.

There are five variants (Debug, ASan, TSan, MSan, UBSan).

## Thread Fuzzer

Thread Fuzzer (please don't mix up with Thread Sanitizer) is another kind of fuzzing that allows to randomize thread order of execution. It helps to find even more special cases.

## Security Audit

People from Yandex Security Team do some basic overview of ClickHouse capabilities from the security standpoint.

## Static Analyzers

We run `clang-tidy` and `PVS-Studio` on per-commit basis. `clang-static-analyzer` checks are also enabled. `clang-tidy` is also used for some style checks.

We have evaluated `clang-tidy`, Coverity, `cppcheck`, `PVS-Studio`, `tscancode`, `CodeQL`. You will find instructions for usage in `tests/instructions/` directory. Also you can read [the article in russian](#).

If you use `CLion` as an IDE, you can leverage some `clang-tidy` checks out of the box.

We also use `shellcheck` for static analysis of shell scripts.

## Hardening

In debug build we are using custom allocator that does ASLR of user-level allocations.

We also manually protect memory regions that are expected to be readonly after allocation.

In debug build we also involve a customization of libc that ensures that no "harmful" (obsolete, insecure, not thread-safe) functions are called.

Debug assertions are used extensively.

In debug build, if exception with "logical error" code (implies a bug) is being thrown, the program is terminated prematurely. It allows to use exceptions in release build but make it an assertion in debug build.

Debug version of jemalloc is used for debug builds.

Debug version of libc++ is used for debug builds.

## Runtime Integrity Checks

Data stored on disk is checksummed. Data in MergeTree tables is checksummed in three ways simultaneously\* (compressed data blocks, uncompressed data blocks, the total checksum across blocks). Data transferred over network between client and server or between servers is also checksummed. Replication ensures bit-identical data on replicas.

It is required to protect from faulty hardware (bit rot on storage media, bit flips in RAM on server, bit flips in RAM of network controller, bit flips in RAM of network switch, bit flips in RAM of client, bit flips on the wire). Note that bit flips are common and likely to occur even for ECC RAM and in presence of TCP checksums (if you manage to run thousands of servers processing petabytes of data each day). [See the video \(russian\)](#).

ClickHouse provides diagnostics that will help ops engineers to find faulty hardware.

\* and it is not slow.

## Code Style

Code style rules are described [here](#).

To check for some common style violations, you can use `utils/check-style` script.

To force proper style of your code, you can use clang-format. File `.clang-format` is located at the sources root. It mostly corresponds with our actual code style. But it's not recommended to apply `clang-format` to existing files because it makes formatting worse. You can use `clang-format-diff` tool that you can find in clang source repository.

Alternatively you can try `unrustify` tool to reformat your code. Configuration is in `unrustify.cfg` in the sources root. It is less tested than `clang-format`.

CLion has its own code formatter that has to be tuned for our code style.

We also use `codespell` to find typos in code. It is automated as well.

## Metrica B2B Tests

Each ClickHouse release is tested with Yandex Metrica and AppMetrica engines. Testing and stable versions of ClickHouse are deployed on VMs and run with a small copy of Metrica engine that is processing fixed sample of input data. Then results of two instances of Metrica engine are compared together.

These tests are automated by separate team. Due to high number of moving parts, tests fail most of the time by completely unrelated reasons, that are very difficult to figure out. Most likely these tests have negative value for us. Nevertheless these tests were proved to be useful in about one or two times out of hundreds.

## Test Coverage

We also track test coverage but only for functional tests and only for clickhouse-server. It is performed on daily basis.

# Tests for Tests

There is automated check for flaky tests. It runs all new tests 100 times (for functional tests) or 10 times (for integration tests). If at least single time the test failed, it is considered flaky.

## Testflows

[Testflows](#) is an enterprise-grade testing framework. It is used by Altinity for some of the tests and we run these tests in our CI.

## Yandex Checks (only for Yandex employees)

These checks are importing ClickHouse code into Yandex internal monorepository, so ClickHouse codebase can be used as a library by other products at Yandex (YT and YDB). Note that clickhouse-server itself is not being build from internal repo and unmodified open-source build is used for Yandex applications.

## Test Automation

We run tests with Yandex internal CI and job automation system named “Sandbox”.

Build jobs and tests are run in Sandbox on per commit basis. Resulting packages and test results are published in GitHub and can be downloaded by direct links. Artifacts are stored for several months. When you send a pull request on GitHub, we tag it as “can be tested” and our CI system will build ClickHouse packages (release, debug, with address sanitizer, etc) for you.

We do not use Travis CI due to the limit on time and computational power.

We do not use Jenkins. It was used before and now we are happy we are not using Jenkins.

## Third-Party Libraries Used

The list of third-party libraries:

Library name	License type
abseil-cpp	Apache
AMQP-CPP	Apache
arrow	Apache
avro	Apache
aws	Apache
aws-c-common	Apache
aws-c-event-stream	Apache
aws-checksums	Apache
base64	BSD 2-clause
boost	Boost

Library name	License type
boringssl	BSD
brotli	MIT
capnproto	MIT
cassandra	Apache
cctz	Apache
cityhash102	MIT
cppkafka	BSD 2-clause
croaring	Apache
curl	Apache
cyrus-sasl	BSD 2-clause
double-conversion	BSD 3-clause
dragonbox	Apache
fast_float	Apache
fastops	MIT
flatbuffers	Apache
fmtlib	Unknown
gcem	Apache
googletest	BSD 3-clause
grpc	Apache
h3	Apache
hyperscan	Boost
icu	Public Domain
icudata	Public Domain
jemalloc	BSD 2-clause
krb5	MIT

Library name	License type
libc-headers	LGPL
libcpuid	BSD 2-clause
libcxx	Apache
libcxxabi	Apache
libdivide	zLib
libfarmhash	MIT
libgsasl	LGPL
libhdfs3	Apache
libmetrohash	Apache
libpq	Unknown
libpqxx	BSD 3-clause
librdkafka	MIT
libunwind	Apache
libuv	BSD
llvm	Apache
lz4	BSD
mariadb-connector-c	LGPL
miniselect	Boost
msgpack-c	Boost
murmurhash	Public Domain
NuRaft	Apache
openldap	Unknown
orc	Apache
poco	Boost
protobuf	BSD 3-clause

Library name	License type
rapidjson	MIT
re2	BSD 3-clause
replxx	BSD 3-clause
rocksdb	BSD 3-clause
s2geometry	Apache
sentry-native	MIT
simdjson	Apache
snappy	Public Domain
sparsehash-c11	BSD 3-clause
stats	Apache
thrift	Apache
unixodbc	LGPL
xz	Public Domain
zlib-ng	zLib
zstd	BSD

The list of third-party libraries can be obtained by the following query:

```
SELECT library_name, license_type, license_path FROM system.licenses ORDER BY library_name COLLATE 'en';
```

#### Example

## Guidelines for adding new third-party libraries and maintaining custom changes in them

1. All external third-party code should reside in the dedicated directories under contrib directory of ClickHouse repo. Prefer Git submodules, when available.
2. Fork/mirror the official repo in [Clickhouse-extras](#). Prefer official GitHub repos, when available.
3. Branch from the branch you want to integrate, e.g., master -> clickhouse/master, or release vX.Y.Z -> clickhouse/release/vX.Y.Z.
4. All forks in [Clickhouse-extras](#) can be automatically synchronized with upstreams. clickhouse/... branches will remain unaffected, since virtually nobody is going to use that naming pattern in their upstream repos.
5. Add submodules under contrib of ClickHouse repo that refer the above forks/mirrors. Set the submodules to track the corresponding clickhouse/... branches.

6. Every time the custom changes have to be made in the library code, a dedicated branch should be created, like clickhouse/my-fix. Then this branch should be merged into the branch, that is tracked by the submodule, e.g., clickhouse/master or clickhouse/release/vX.Y.Z.
7. No code should be pushed in any branch of the forks in **Clickhouse-extras**, whose names do not follow clickhouse/... pattern.
8. Always write the custom changes with the official repo in mind. Once the PR is merged from (a feature/fix branch in) your personal fork into the fork in **Clickhouse-extras**, and the submodule is bumped in ClickHouse repo, consider opening another PR from (a feature/fix branch in) the fork in **Clickhouse-extras** to the official repo of the library. This will make sure, that 1) the contribution has more than a single use case and importance, 2) others will also benefit from it, 3) the change will not remain a maintenance burden solely on ClickHouse developers.
9. When a submodule needs to start using a newer code from the original branch (e.g., master), and since the custom changes might be merged in the branch it is tracking (e.g., clickhouse/master) and so it may diverge from its original counterpart (i.e., master), a careful merge should be carried out first, i.e., master -> clickhouse/master, and only then the submodule can be bumped in ClickHouse.

## Browse ClickHouse Source Code

You can use **Woboq** online code browser available [here](#). It provides code navigation and semantic highlighting, search and indexing. The code snapshot is updated daily.

Also, you can browse sources on [GitHub](#) as usual.

If you're interested what IDE to use, we recommend CLion, QT Creator, VS Code and KDevelop (with caveats). You can use any favourite IDE. Vim and Emacs also count.

## CMake in ClickHouse

### TL; DR How to make ClickHouse compile and link faster?

Minimal ClickHouse build example:

```
cmake .. \
-DCMAKE_C_COMPILER=$(which clang-11) \
-DCMAKE_CXX_COMPILER=$(which clang++-11) \
-DCMAKE_BUILD_TYPE=Debug \
-DENABLE_CLICKHOUSE_ALL=OFF \
-ENABLE_CLICKHOUSE_SERVER=ON \
-ENABLE_CLICKHOUSE_CLIENT=ON \
-ENABLE_LIBRARIES=OFF \
-DUSE_UNWIND=ON \
-ENABLE_UTILS=OFF \
-ENABLE_TESTS=OFF
```

## CMake files types

1. ClickHouse's source CMake files (located in the root directory and in /src).
2. Arch-dependent CMake files (located in /cmake/\*os\_name\*).
3. Libraries finders (search for contrib libraries, located in /cmake/find).
4. Contrib build CMake files (used instead of libraries' own CMake files, located in /cmake/modules)

## List of CMake flags

- This list is auto-generated by [this Python script](#).
- The flag name is a link to its position in the code.

- If an option's default value is itself an option, it's also a link to its position in this list.

## ClickHouse modes

Name	Default value	Description	Comment
ENABLE_CLICKHOUSE_ALL	ON	Enable all ClickHouse modes by default	The clickhouse tool multiple executors may be built a know what mode SERVER and CLIENT
ENABLE_CLICKHOUSE_BENCHMARK	ENABLE_CLICKHOUSE_ALL	Queries benchmarking mode	<a href="https://clickhouse.com/docs/en/interfaces/benchmark/">https://clickhouse.com/docs/en/interfaces/benchmark/</a>
ENABLE_CLICKHOUSE_CLIENT	ENABLE_CLICKHOUSE_ALL	Client mode (interactive tui/shell that connects to the server)	
ENABLE_CLICKHOUSE_COMPRESSOR	ENABLE_CLICKHOUSE_ALL	Data compressor and decompressor	<a href="https://clickhouse.com/docs/en/interfaces/compressor/">https://clickhouse.com/docs/en/interfaces/compressor/</a>
ENABLE_CLICKHOUSE_COPIER	ENABLE_CLICKHOUSE_ALL	Inter-cluster data copying mode	<a href="https://clickhouse.com/docs/en/interfaces/copier/">https://clickhouse.com/docs/en/interfaces/copier/</a>
ENABLE_CLICKHOUSE_EXTRACT_FROM_CONFIG	ENABLE_CLICKHOUSE_ALL	Configs processor (extract values etc.)	
ENABLE_CLICKHOUSE_FORMAT	ENABLE_CLICKHOUSE_ALL	Queries pretty-printer and formatter with syntax highlighting	
ENABLE_CLICKHOUSE_GIT_IMPORT	ENABLE_CLICKHOUSE_ALL	A tool to analyze Git repositories	<a href="https://presentations.clickhouse.com/presentations/analyze-git-repositories">https://presentations.clickhouse.com/presentations/analyze-git-repositories</a>
ENABLE_CLICKHOUSE_INSTALL	OFF	Install ClickHouse without .deb/.rpm/.tgz packages (having the binary only)	

Name	Default value	Description	Comment
ENABLE_CLICKHOUSE_KEEPER	ENABLE_CLICKHOUSE_ALL	ClickHouse alternative to ZooKeeper	
ENABLE_CLICKHOUSE_KEEPER_CONVERTER	ENABLE_CLICKHOUSE_ALL	Util allows to convert ZooKeeper logs and snapshots into clickhouse-keeper snapshot	
ENABLE_CLICKHOUSE_LIBRARY_BRIDGE	ENABLE_CLICKHOUSE_ALL	HTTP-server working like a proxy to Library dictionary source	
ENABLE_CLICKHOUSE_LOCAL	ENABLE_CLICKHOUSE_ALL	Local files fast processing mode	<a href="https://clickho...local/">https://clickho...local/</a>
ENABLE_CLICKHOUSE_OBFUSCATOR	ENABLE_CLICKHOUSE_ALL	Table data obfuscator (convert real data to benchmark-ready one)	<a href="https://clickho...obfuscator/">https://clickho...obfuscator/</a>
ENABLE_CLICKHOUSE_ODBC_BRIDGE	ENABLE_CLICKHOUSE_ALL	HTTP-server working like a proxy to ODBC driver	
ENABLE_CLICKHOUSE_SERVER	ENABLE_CLICKHOUSE_ALL	Server mode (main mode)	
ENABLE_CLICKHOUSE_STATIC_FILES_DISK_UPLOADER	ENABLE_CLICKHOUSE_ALL	A tool to export table data files to be later put to a static files web server	

## External libraries

Note that ClickHouse uses forks of these libraries, see <https://github.com/ClickHouse-Extras>.

Name	Default value	Description	Comment
ENABLE_AMQPCPP	ENABLE_LIBRARIES	Enable AMQP-CPP	
ENABLE_AVRO	ENABLE_LIBRARIES	Enable Avro	Needed when serializing
ENABLE_AVX	0	Use AVX instructions on x86_64	
ENABLE_AVX2	0	Use AVX2 instructions on x86_64	
ENABLE_AVX2_FOR_SPEC_OP	0	Use avx2 instructions for specific operations on x86_64	
ENABLE_AVX512	0	Use AVX512 instructions on x86_64	
ENABLE_AVX512_FOR_SPEC_OP	0	Use avx512 instructions for specific operations on x86_64	
ENABLE_BASE64	ENABLE_LIBRARIES	Enable base64	
ENABLE_BMI	0	Use BMI instructions on x86_64	
ENABLE_BROTLI	ENABLE_LIBRARIES	Enable brotli	
ENABLE_BZIP2	ENABLE_LIBRARIES	Enable bzip2 compression support	
ENABLE_CAPNP	ENABLE_LIBRARIES	Enable Cap'n Proto	
ENABLE_CASSANDRA	ENABLE_LIBRARIES	Enable Cassandra	
ENABLE_CCACHE	ENABLE_CCACHE_BY_DEFAULT	Speedup re-compilations using ccache (external tool)	<a href="https://ccache.suckless.org">https://ccache.suckless.org</a>
ENABLE_CLANG_TIDY	OFF	Use clang-tidy static analyzer	<a href="https://clang-tidy.readthedocs.io/">https://clang-tidy.readthedocs.io/</a>
ENABLE_CURL	ENABLE_LIBRARIES	Enable curl	
ENABLE_DATASKETCHES	ENABLE_LIBRARIES	Enable DataSketches	

Name	Default value	Description	Comment
ENABLE_EMBEDDED_COMPILER	ENABLE_EMBEDDED_COMPILER_DEFAULT	Enable support for 'compile_expressions' option for query execution	
ENABLE_FASTOPS	ENABLE_LIBRARIES	Enable fast vectorized mathematical functions library by Mikhail Parakhin	
ENABLE_FILELOG	ON	Enable FILELOG	
ENABLE_GPERF	ENABLE_LIBRARIES	Use gperf function hash generator tool	
ENABLE_GRPC	ENABLE_GRPC_DEFAULT	Use gRPC	
ENABLE_GSASL_LIBRARY	ENABLE_LIBRARIES	Enable gsasl library	
ENABLE_H3	ENABLE_LIBRARIES	Enable H3	
ENABLE_HDFS	ENABLE_LIBRARIES	Enable HDFS	
ENABLE_ICU	ENABLE_LIBRARIES	Enable ICU	
ENABLE_LDAP	ENABLE_LIBRARIES	Enable LDAP	
ENABLE_LIBPQXX	ENABLE_LIBRARIES	Enable libpqxx	
ENABLE_MSGPACK	ENABLE_LIBRARIES	Enable msgpack library	
ENABLE_MYSQL	ENABLE_LIBRARIES	Enable MySQL	
ENABLE_NLP	ENABLE_LIBRARIES	Enable NLP functions support	
ENABLE_NURAFFT	ENABLE_LIBRARIES	Enable NuRaft	
ENABLE_ODBC	ENABLE_LIBRARIES	Enable ODBC library	
ENABLE_ORC	ENABLE_LIBRARIES	Enable ORC	
ENABLE_PARQUET	ENABLE_LIBRARIES	Enable parquet	
ENABLE_PCLMULQDQ	1	Use pclmulqdq instructions on x86_64	

Name	Default value	Description	Comment
ENABLE_POPCNT	1	Use popcnt instructions on x86_64	
ENABLE_PROTOBUF	ENABLE_LIBRARIES	Enable protobuf	
ENABLE_RAPIDJSON	ENABLE_LIBRARIES	Use rapidjson	
ENABLE_RDKAFKA	ENABLE_LIBRARIES	Enable kafka	
ENABLE_ROCKSDB	ENABLE_LIBRARIES	Enable ROCKSDB	
ENABLE_S2_GEOMETRY	ENABLE_LIBRARIES	Enable S2 geometry library	
ENABLE_S3	ENABLE_LIBRARIES	Enable S3	
ENABLE_SQLITE	ENABLE_LIBRARIES	Enable sqlite	
ENABLE_SSE41	1	Use SSE4.1 instructions on x86_64	
ENABLE_SSE42	1	Use SSE4.2 instructions on x86_64	
ENABLE_SSL	ENABLE_LIBRARIES	Enable ssl	Needed when connecting to e.g. clickhouse secure
ENABLE_SSSE3	1	Use SSSE3 instructions on x86_64	
ENABLE_STATS	ENABLE_LIBRARIES	Enable StatsLib library	

## External libraries system/bundled mode

Name	Default value	Description	Comment
USE_INTERNAL_AVRO_LIBRARY	ON	Set to FALSE to use system avro library instead of bundled	

Name	Default value	Description	Comment
USE_INTERNAL_AWS_S3_LIBRARY	ON	Set to FALSE to use system S3 instead of bundled (experimental set to OFF on your own risk)	
USE_INTERNAL_BROTLI_LIBRARY	USE_STATIC_LIBRARIES	Set to FALSE to use system libbrotli library instead of bundled	Many system brotly libraries bundled by
USE_INTERNAL_CAPNP_LIBRARY	NOT_UNBUNDLED	Set to FALSE to use system capnproto library instead of bundled	
USE_INTERNAL_CURL	NOT_UNBUNDLED	Use internal curl library	
USE_INTERNAL_DATASKETCHES_LIBRARY	NOT_UNBUNDLED	Set to FALSE to use system DataSketches library instead of bundled	
USE_INTERNAL_GRPC_LIBRARY	NOT_UNBUNDLED	Set to FALSE to use system gRPC library instead of bundled. (Experimental. Set to OFF on your own risk)	Normally we framework USE_INTERNAL_GPRC set to OFF to force gRPC framework installed in The external be installed running sudo libgrpc++- grpc
USE_INTERNAL_GTEST_LIBRARY	NOT_UNBUNDLED	Set to FALSE to use system Google Test instead of bundled	

Name	Default value	Description	Comment
USE_INTERNAL_H3_LIBRARY	ON	Set to FALSE to use system h3 library instead of bundled	
USE_INTERNAL_HDFS3_LIBRARY	ON	Set to FALSE to use system HDFS3 instead of bundled (experimental - set to OFF on your own risk)	
USE_INTERNAL_ICU_LIBRARY	NOT_UNBUNDLED	Set to FALSE to use system ICU library instead of bundled	
USE_INTERNAL_LDAP_LIBRARY	NOT_UNBUNDLED	Set to FALSE to use system *LDAP library instead of bundled	
USE_INTERNAL_LIBCXX_LIBRARY	USE_INTERNAL_LIBCXX_LIBRARY_DEFAULT	Disable to use system libcxx and libcxxabi libraries instead of bundled	
USE_INTERNAL_LIBGSSASL_LIBRARY	USE_STATIC_LIBRARIES	Set to FALSE to use system libgsasl library instead of bundled	when USE_ usually nee dependenc
USE_INTERNAL_LIBXML2_LIBRARY	NOT_UNBUNDLED	Set to FALSE to use system libxml2 library instead of bundled	
USE_INTERNAL_MSGPACK_LIBRARY	NOT_UNBUNDLED	Set to FALSE to use system msgpack library instead of bundled	

Name	Default value	Description	Comment
USE_INTERNAL_MYSQL_LIBRARY	NOT_UNBUNDLED	Set to FALSE to use system mysqlclient library instead of bundled	
USE_INTERNAL_ODBC_LIBRARY	NOT_UNBUNDLED	Use internal ODBC library	
USE_INTERNAL_ORC_LIBRARY	ON	Set to FALSE to use system ORC instead of bundled (experimental set to OFF on your own risk)	
USE_INTERNAL_PARQUET_LIBRARY	NOT_UNBUNDLED	Set to FALSE to use system parquet library instead of bundled	
USE_INTERNAL_POCO_LIBRARY	ON	Use internal Poco library	
USE_INTERNAL_PROTOBUF_LIBRARY	NOT_UNBUNDLED	Set to FALSE to use system protobuf instead of bundled. (Experimental. Set to OFF on your own risk)	Normally w USE_INTER to OFF to fo protobuf lib installed in The exten installed in sudo apt-g protobuf-cc
USE_INTERNAL_RAPIDJSON_LIBRARY	NOT_UNBUNDLED	Set to FALSE to use system rapidjson library instead of bundled	
USE_INTERNAL_RDKAFKA_LIBRARY	NOT_UNBUNDLED	Set to FALSE to use system librdkafka instead of the bundled	

Name	Default value	Description	Comment
USE_INTERNAL_RE2_LIBRARY	NOT_UNBUNDLED	Set to FALSE to use system re2 library instead of bundled [slower]	
USE_INTERNAL_ROCKSDB_LIBRARY	NOT_UNBUNDLED	Set to FALSE to use system ROCKSDB library instead of bundled	
USE_INTERNAL_SNAPPY_LIBRARY	NOT_UNBUNDLED	Set to FALSE to use system snappy library instead of bundled	
USE_INTERNAL_SPARSEHASH_LIBRARY	ON	Set to FALSE to use system sparsehash library instead of bundled	
USE_INTERNAL_SSL_LIBRARY	NOT_UNBUNDLED	Set to FALSE to use system *ssl library instead of bundled	
USE_INTERNAL_XZ_LIBRARY	NOT_UNBUNDLED	Set to OFF to use system xz (lzma) library instead of bundled	
USE_INTERNAL_ZLIB_LIBRARY	NOT_UNBUNDLED	Set to FALSE to use system zlib library instead of bundled	
USE_INTERNAL_ZSTD_LIBRARY	NOT_UNBUNDLED	Set to FALSE to use system zstd library instead of bundled	

## Other flags

Name	Default value	Description	Comment
ADD_GDB_INDEX_FOR_GOLD	OFF	Add .gdb-index to resulting binaries for gold linker.	Ignored if <code>lld</code> is used
ARCH_NATIVE	0	Add <code>-march=native</code> compiler flag. This makes your binaries non-portable but more performant code may be generated. This option overrides <code>ENABLE_*</code> options for specific instruction set. Highly not recommended to use.	
CLICKHOUSE_SPLIT_BINARY	OFF	Make several binaries (clickhouse-server, clickhouse-client etc.) instead of one bundled	
COMPILER_PIPE	ON	-pipe compiler option	Less <code>/tmp</code> usage, mc



Name	Default value	Description	Comment
FAIL_ON_UNSUPPORTED_OPTIONS_COMBINATION	ON	Stop/Fail CMake configuration if some ENABLE_XXX option is defined (either ON or OFF) but is not possible to satisfy	If turned off: e.g. w/o the CMake will continue
GLIBC_COMPATIBILITY	ON	Enable compatibility with older glibc libraries.	Only for Linux, x86_64
LINKER_NAME	OFF	Linker name or full path	Example values: lld-link
MAKE_STATIC_LIBRARIES	USE_STATIC_LIBRARIES	Disable to make shared libraries	
PARALLEL_COMPILE_JOBS	""	Maximum number of concurrent compilation jobs	1 if not set
PARALLEL_LINK_JOBS	""	Maximum number of concurrent link jobs	1 if not set
SANITIZE	""	Enable one of the code sanitizers	Possible values: -fno-undefined (UBSan) -fno-sanitize=*
SPLIT_SHARED_LIBRARIES	OFF	Keep all internal libraries as separate .so files	DEVELOPER ONLY. If
STRIP_DEBUG_SYMBOLS_FUNCTIONS	STRIP_DSF_DEFAULT	Do not generate debugger info for ClickHouse functions	Provides faster linking (without debug some source variables)."

Name	Default value	Description	Comment
UNBUNDLED	OFF	Use system libraries instead of ones in contrib/	We recommend avc can't guarantee all i exists for enthusiast whole idea of using is deeply flawed. Us
USE_INCLUDE_WHAT_YOU_USE	OFF	Automatically reduce unneeded includes in source code (external tool)	<a href="https://github.com/i">https://github.com/i</a>
USE_LIBCXX	NOT_UNBUNDLED	Use libc++ and libc++abi instead of libstdc++	
USE_LIBPROTOBUF_MUTATOR	ENABLE_FUZZING	Enable libprotobuf-mutator	
USE_SENTRY	ENABLE_LIBRARIES	Use Sentry	
USE_SIMDJSON	ENABLE_LIBRARIES	Use simdjson	
USE_SNAPPY	ENABLE_LIBRARIES	Enable snappy library	
USE_STATIC_LIBRARIES	ON	Disable to use shared libraries	
USE_UNWIND	ENABLE_LIBRARIES	Enable libunwind (better stacktraces)	
USE_YAML_CPP	ENABLE_LIBRARIES	Enable yaml-cpp	
WERROR	OFF	Enable -Werror compiler option	Using system libs can expand).
WEVERYTHING	ON	Enable -Weverything option with some exceptions.	Add some warnings Wpedantic. Intended to be found useful. API

Name	Default value	Description	Comment
WITH_COVERAGE	OFF	Profile the resulting binary/binaries	Compiler-specific cc

## Developer's guide for adding new CMake options

Don't be obvious. Be informative.

Bad:

```
option(ENABLE_TESTS "Enables testing" OFF)
```

This description is quite useless as it neither gives the viewer any additional information nor explains the option purpose.

Better:

```
option(ENABLE_TESTS "Provide unit_test_dbms target with Google.test unit tests" OFF)
```

If the option's purpose can't be guessed by its name, or the purpose guess may be misleading, or option has some

pre-conditions, leave a comment above the option() line and explain what it does.

The best way would be linking the docs page (if it exists).

The comment is parsed into a separate column (see below).

Even better:

```
## implies ${TESTS_ARE_ENABLED}
## see tests/CMakeLists.txt for implementation detail.
option(ENABLE_TESTS "Provide unit_test_dbms target with Google.test unit tests" OFF)
```

If the option's state could produce unwanted (or unusual) result, explicitly warn the user.

Suppose you have an option that may strip debug symbols from the ClickHouse's part.

This can speed up the linking process, but produces a binary that cannot be debugged.

In that case, prefer explicitly raising a warning telling the developer that he may be doing something wrong.

Also, such options should be disabled if applies.

Bad:

```
option(STRIPE_DEBUG_SYMBOLS_FUNCTIONS
    "Do not generate debugger info for ClickHouse functions.
    ${STRIPE_DSF_DEFAULT}")

if (STRIPE_DEBUG_SYMBOLS_FUNCTIONS)
    target_compile_options(clickhouse_functions PRIVATE "-g0")
endif()
```

Better:

```

## Provides faster linking and lower binary size.
## Tradeoff is the inability to debug some source files with e.g. gdb
## (empty stack frames and no local variables)."
option(STRIPE_DEBUG_SYMBOLS_FUNCTIONS
    "Do not generate debugger info for ClickHouse functions."
    ${STRIP_DSF_DEFAULT})

if (STRIPE_DEBUG_SYMBOLS_FUNCTIONS)
    message(WARNING "Not generating debugger info for ClickHouse functions")
    target_compile_options(clickhouse_functions PRIVATE "-g0")
endif()

```

## In the option's description, explain WHAT the option does rather than WHY it does something.

The WHY explanation should be placed in the comment.

You may find that the option's name is self-descriptive.

Bad:

```
option(ENABLE_THINLTO "Enable Thin LTO. Only applicable for clang. It's also suppressed when building with tests or sanitizers." ON)
```

Better:

```

## Only applicable for clang.
## Turned off when building with tests or sanitizers.
option(ENABLE_THINLTO "Clang-specific link time optimisation" ON).

```

## Don't assume other developers know as much as you do.

In ClickHouse, there are many tools used that an ordinary developer may not know. If you are in doubt, give a link to the tool's docs. It won't take much of your time.

Bad:

```
option(ENABLE_THINLTO "Enable Thin LTO. Only applicable for clang. It's also suppressed when building with tests or sanitizers." ON)
```

Better (combined with the above hint):

```

## https://clang.llvm.org/docs/ThinLTO.html
## Only applicable for clang.
## Turned off when building with tests or sanitizers.
option(ENABLE_THINLTO "Clang-specific link time optimisation" ON).

```

Other example, bad:

```
option (USE_INCLUDE_WHAT_YOU_USE "Use 'include-what-you-use' tool" OFF)
```

Better:

```

## https://github.com/include-what-you-use/include-what-you-use
option (USE_INCLUDE_WHAT_YOU_USE "Reduce unneeded #include s (external tool)" OFF)

```

## Prefer consistent default values.

CMake allows you to pass a plethora of values representing boolean true/false, e.g. 1, ON, YES, ....  
Prefer the ON/OFF values, if possible.

# How to add test queries to ClickHouse CI

ClickHouse has hundreds (or even thousands) of features. Every commit gets checked by a complex set of tests containing many thousands of test cases.

The core functionality is very well tested, but some corner-cases and different combinations of features can be uncovered with ClickHouse CI.

Most of the bugs/regressions we see happen in that 'grey area' where test coverage is poor.

And we are very interested in covering most of the possible scenarios and feature combinations used in real life by tests.

## Why adding tests

Why/when you should add a test case into ClickHouse code:

- 1) you use some complicated scenarios / feature combinations / you have some corner case which is probably not widely used
- 2) you see that certain behavior gets changed between version w/o notifications in the changelog
- 3) you just want to help to improve ClickHouse quality and ensure the features you use will not be broken in the future releases
- 4) once the test is added/accepted, you can be sure the corner case you check will never be accidentally broken.
- 5) you will be a part of great open-source community
- 6) your name will be visible in the system.contributors table!
- 7) you will make a world bit better :)

## Steps to do

### Prerequisite

I assume you run some Linux machine (you can use docker / virtual machines on other OS) and any modern browser / internet connection, and you have some basic Linux & SQL skills.

Any highly specialized knowledge is not needed (so you don't need to know C++ or know something about how ClickHouse CI works).

### Preparation

- 1) [create GitHub account](#) (if you haven't one yet)
- 2) [setup git](#)

```
## for Ubuntu
sudo apt-get update
sudo apt-get install git

git config --global user.name "John Doe" # fill with your name
git config --global user.email "email@example.com" # fill with your email
```

3) fork ClickHouse project - just open <https://github.com/ClickHouse/ClickHouse> and press fork button in the top right corner:



4) clone your fork to some folder on your PC, for example, ~/workspace/ClickHouse

```
mkdir ~/workspace && cd ~/workspace
git clone https://github.com/< your GitHub username>/ClickHouse
cd ClickHouse
git remote add upstream https://github.com/ClickHouse/ClickHouse
```

New branch for the test

1) create a new branch from the latest clickhouse master

```
cd ~/workspace/ClickHouse
git fetch upstream
git checkout -b name_for_a_branch_with_my_test upstream/master
```

Install & run clickhouse

1) install clickhouse-server (follow [official docs](#))  
2) install test configurations (it will use Zookeeper mock implementation and adjust some settings)

```
cd ~/workspace/ClickHouse/tests/config
sudo ./install.sh
```

3) run clickhouse-server

```
sudo systemctl restart clickhouse-server
```

Creating the test file

1) find the number for your test - find the file with the biggest number in tests/queries/0\_stateless/

```
$ cd ~/workspace/ClickHouse
$ ls tests/queries/0_stateless/[0-9]*.reference | tail -n 1
tests/queries/0_stateless/01520_client_print_query_id.reference
```

Currently, the last number for the test is 01520, so my test will have the number 01521

2) create an SQL file with the next number and name of the feature you test

```
touch tests/queries/0_stateless/01521_dummy_test.sql
```

3) edit SQL file with your favorite editor (see hint of creating tests below)

```
vim tests/queries/0_stateless/01521_dummy_test.sql
```

4) run the test, and put the result of that into the reference file:

```
clickhouse-client -nmT < tests/queries/0_stateless/01521_dummy_test.sql | tee  
tests/queries/0_stateless/01521_dummy_test.reference
```

5) ensure everything is correct, if the test output is incorrect (due to some bug for example), adjust the reference file using text editor.

## How to create a good test

- A test should be
  - minimal - create only tables related to tested functionality, remove unrelated columns and parts of query
  - fast - should not take longer than a few seconds (better subseconds)
  - correct - fails then feature is not working
    - deterministic
  - isolated / stateless
    - don't rely on some environment things
    - don't rely on timing when possible
- try to cover corner cases (zeros / Nulls / empty sets / throwing exceptions)
- to test that query return errors, you can put special comment after the query: `-- { serverError 60 }` or `-- { clientError 20 }`
- don't switch databases (unless necessary)
- you can create several table replicas on the same node if needed
- you can use one of the test cluster definitions when needed (see `system.clusters`)
- use `number` / `numbers_mt` / `zeros` / `zeros_mt` and similar for queries / to initialize data when applicable
- clean up the created objects after test and before the test (DROP IF EXISTS) - in case of some dirty state
- prefer sync mode of operations (mutations, merges, etc.)
- use other SQL files in the `0_stateless` folder as an example
- ensure the feature / feature combination you want to test is not yet covered with existing tests

## Test naming rules

It's important to name tests correctly, so one could turn some tests subset off in `clickhouse-test` invocation.

Tester flag	What should be in test name	When flag should be added
--- --- --- ---		
--[no-]zookeeper	"zookeeper" or "replica"	Test uses tables from ReplicatedMergeTree family
--[no-]shard	"shard" or "distributed" or "global"	Test using connections to 127.0.0.2 or similar
--[no-]long	"long" or "deadlock" or "race"	Test runs longer than 60 seconds

Commit / push / create PR.

- 1) commit & push your changes

```
cd ~/workspace/ClickHouse  
git add tests/queries/0_stateless/01521_dummy_test.sql  
git add tests/queries/0_stateless/01521_dummy_test.reference  
git commit # use some nice commit message when possible  
git push origin HEAD
```

- 2) use a link which was shown during the push, to create a PR into the main repo
- 3) adjust the PR title and contents, in Changelog category (leave one) keep Build/Testing/Packaging Improvement, fill the rest of the fields if you want.

## ClickHouse Cloud Service

### Info

Detailed public description for ClickHouse cloud services is not ready yet, please [contact us](#) to learn more.

## ClickHouse Commercial Support Service

### Info

Detailed public description for ClickHouse support services is not ready yet, please [contact us](#) to learn more.

## ClickHouse Commercial Services

Service categories:

- [Cloud](#)
- [Support](#)

## General Questions About ClickHouse

Questions:

- [What is ClickHouse?](#)
- [Why ClickHouse is so fast?](#)
- [Who is using ClickHouse?](#)
- [What does “ClickHouse” mean?](#)
- [What does “He тормозит” mean?](#)
- [What is OLAP?](#)
- [What is a columnar database?](#)
- [Why not use something like MapReduce?](#)

## Don't see what you were looking for?

Check out [other F.A.Q. categories](#) or browse around main documentation articles found in the left sidebar.

# Why ClickHouse Is So Fast?

It was designed to be fast. Query execution performance has always been a top priority during the development process, but other important characteristics like user-friendliness, scalability, and security were also considered so ClickHouse could become a real production system.

ClickHouse was initially built as a prototype to do just a single task well: to filter and aggregate data as fast as possible. That's what needs to be done to build a typical analytical report and that's what a typical [GROUP BY](#) query does. ClickHouse team has made several high-level decisions that combined made achieving this task possible:

### Column-oriented storage

Source data often contain hundreds or even thousands of columns, while a report can use just a few of them. The system needs to avoid reading unnecessary columns, or most expensive disk read operations would be wasted.

### Indexes

ClickHouse keeps data structures in memory that allows reading not only used columns but only necessary row ranges of those columns.

### Data compression

Storing different values of the same column together often leads to better compression ratios (compared to row-oriented systems) because in real data column often has the same or not so many different values for neighboring rows. In addition to general-purpose compression, ClickHouse supports [specialized codecs](#) that can make data even more compact.

### Vectorized query execution

ClickHouse not only stores data in columns but also processes data in columns. It leads to better CPU cache utilization and allows for [SIMD](#) CPU instructions usage.

### Scalability

ClickHouse can leverage all available CPU cores and disks to execute even a single query. Not only on a single server but all CPU cores and disks of a cluster as well.

But many other database management systems use similar techniques. What really makes ClickHouse stand out is **attention to low-level details**. Most programming languages provide implementations for most common algorithms and data structures, but they tend to be too generic to be effective. Every task can be considered as a landscape with various characteristics, instead of just throwing in random implementation. For example, if you need a hash table, here are some key questions to consider:

- Which hash function to choose?
- Collision resolution algorithm: [open addressing](#) vs [chaining](#)?
- Memory layout: one array for keys and values or separate arrays? Will it store small or large values?
- Fill factor: when and how to resize? How to move values around on resize?
- Will values be removed and which algorithm will work better if they will?
- Will we need fast probing with bitmaps, inline placement of string keys, support for non-movable values, prefetch, and batching?

Hash table is a key data structure for GROUP BY implementation and ClickHouse automatically chooses one of [30+ variations](#) for each specific query.

The same goes for algorithms, for example, in sorting you might consider:

- What will be sorted: an array of numbers, tuples, strings, or structures?
- Is all data available completely in RAM?
- Do we need a stable sort?
- Do we need a full sort? Maybe partial sort or n-th element will suffice?
- How to implement comparisons?
- Are we sorting data that has already been partially sorted?

Algorithms that they rely on characteristics of data they are working with can often do better than their generic counterparts. If it is not really known in advance, the system can try various implementations and choose the one that works best in runtime. For example, see an [article on how LZ4 decompression is implemented in ClickHouse](#).

Last but not least, the ClickHouse team always monitors the Internet on people claiming that they came up with the best implementation, algorithm, or data structure to do something and tries it out. Those claims mostly appear to be false, but from time to time you'll indeed find a gem.

## Tips for building your own high-performance software

- Keep in mind low-level details when designing your system.
- Design based on hardware capabilities.
- Choose data structures and abstractions based on the needs of the task.
- Provide specializations for special cases.
- Try new, “best” algorithms, that you read about yesterday.
- Choose an algorithm in runtime based on statistics.
- Benchmark on real datasets.
- Test for performance regressions in CI.
- Measure and observe everything.

## Who Is Using ClickHouse?

Being an open-source product makes this question not so straightforward to answer. You do not have to tell anyone if you want to start using ClickHouse, you just go grab source code or pre-compiled packages. There's no contract to sign and the [Apache 2.0 license](#) allows for unconstrained software distribution.

Also, the technology stack is often in a grey zone of what's covered by an NDA. Some companies consider technologies they use as a competitive advantage even if they are open-source and do not allow employees to share any details publicly. Some see some PR risks and allow employees to share implementation details only with their PR department approval.

So how to tell who is using ClickHouse?

One way is to **ask around**. If it's not in writing, people are much more willing to share what technologies are used in their companies, what the use cases are, what kind of hardware is used, data volumes, etc. We're talking with users regularly on [ClickHouse Meetups](#) all over the world and have heard stories about 1000+ companies that use ClickHouse. Unfortunately, that's not reproducible and we try to treat such stories as if they were told under NDA to avoid any potential troubles. But you can come to any of our future meetups and talk with other users on your own. There are multiple ways how meetups are announced, for example, you can subscribe to [our Twitter](#).

The second way is to look for companies **publicly saying** that they use ClickHouse. It's more substantial because there's usually some hard evidence like a blog post, talk video recording, slide deck, etc. We collect the collection of links to such evidence on our [Adopters](#) page. Feel free to contribute the story of your employer or just some links you've stumbled upon (but try not to violate your NDA in the process).

You can find names of very large companies in the adopters list, like Bloomberg, Cisco, China Telecom, Tencent, or Uber, but with the first approach, we found that there are many more. For example, if you take [the list of largest IT companies by Forbes \(2020\)](#) over half of them are using ClickHouse in some way. Also, it would be unfair not to mention [Yandex](#), the company which initially open-sourced ClickHouse in 2016 and happens to be one of the largest IT companies in Europe.

## What Does “ClickHouse” Mean?

It's a combination of “**Clickstream**” and “**Data wareHouse**”. It comes from the original use case at Yandex.Metrica, where ClickHouse was supposed to keep records of all clicks by people from all over the Internet, and it still does the job. You can read more about this use case on [ClickHouse history](#) page.

This two-part meaning has two consequences:

- The only correct way to write ClickHouse is with capital H.
- If you need to abbreviate it, use **CH**. For some historical reasons, abbreviating as CK is also popular in China, mostly because one of the first talks about ClickHouse in Chinese used this form.

### Fun fact

Many years after ClickHouse got its name, this approach of combining two words that are meaningful on their own has been highlighted as the best way to name a database in a [research by Andy Pavlo](#), an Associate Professor of Databases at Carnegie Mellon University. ClickHouse shared his “best database name of all time” award with Postgres.

## What Does “Не тормозит” Mean?

This question usually arises when people see official ClickHouse t-shirts. They have large words **“ClickHouse не тормозит”** on the front.

Before ClickHouse became open-source, it has been developed as an in-house storage system by the largest Russian IT company, [Yandex](#). That's why it initially got its slogan in Russian, which is “не тормозит” (pronounced as “ne tormozit”). After the open-source release we first produced some of those t-shirts for events in Russia and it was a no-brainer to use the slogan as-is.

One of the following batches of those t-shirts was supposed to be given away on events outside of Russia and we tried to make the English version of the slogan. Unfortunately, the Russian language is kind of elegant in terms of expressing stuff and there was a restriction of limited space on a t-shirt, so we failed to come up with good enough translation (most options appeared to be either long or inaccurate) and decided to keep the slogan in Russian even on t-shirts produced for international events. It appeared to be a great decision because people all over the world get positively surprised and curious when they see it.

So, what does it mean? Here are some ways to translate “*не тормозит*”:

- If you translate it literally, it'd be something like “*ClickHouse does not press the brake pedal*”.
- If you'd want to express it as close to how it sounds to a Russian person with IT background, it'd be something like “*If your larger system lags, it's not because it uses ClickHouse*”.
- Shorter, but not so precise versions could be “*ClickHouse is not slow*”, “*ClickHouse does not lag*” or just “*ClickHouse is fast*”.

If you haven't seen one of those t-shirts in person, you can check them out online in many ClickHouse-related videos. For example, this one:

---

P.S. These t-shirts are not for sale, they are given away for free on most [ClickHouse Meetups](#), usually for best questions or other forms of active participation.

## What Is OLAP?

**OLAP** stands for Online Analytical Processing. It is a broad term that can be looked at from two perspectives: technical and business. But at the very high level, you can just read these words backward:

### **Processing**

Some source data is processed...

### **Analytical**

...to produce some analytical reports and insights...

### **Online**

...in real-time.

# OLAP from the Business Perspective

In recent years, business people started to realize the value of data. Companies who make their decisions blindly, more often than not fail to keep up with the competition. The data-driven approach of successful companies forces them to collect all data that might be remotely useful for making business decisions and need mechanisms to timely analyze them. Here's where OLAP database management systems (DBMS) come in.

In a business sense, OLAP allows companies to continuously plan, analyze, and report operational activities, thus maximizing efficiency, reducing expenses, and ultimately conquering the market share. It could be done either in an in-house system or outsourced to SaaS providers like web/mobile analytics services, CRM services, etc. OLAP is the technology behind many BI applications (Business Intelligence).

ClickHouse is an OLAP database management system that is pretty often used as a backend for those SaaS solutions for analyzing domain-specific data. However, some businesses are still reluctant to share their data with third-party providers and an in-house data warehouse scenario is also viable.

## OLAP from the Technical Perspective

All database management systems could be classified into two groups: OLAP (Online **Analytical** Processing) and OLTP (Online **Transactional** Processing). Former focuses on building reports, each based on large volumes of historical data, but doing it not so frequently. While the latter usually handle a continuous stream of transactions, constantly modifying the current state of data.

In practice OLAP and OLTP are not categories, it's more like a spectrum. Most real systems usually focus on one of them but provide some solutions or workarounds if the opposite kind of workload is also desired. This situation often forces businesses to operate multiple storage systems integrated, which might be not so big deal but having more systems make it more expensive to maintain. So the trend of recent years is HTAP (**Hybrid Transactional/Analytical Processing**) when both kinds of the workload are handled equally well by a single database management system.

Even if a DBMS started as a pure OLAP or pure OLTP, they are forced to move towards that HTAP direction to keep up with their competition. And ClickHouse is no exception, initially, it has been designed as **fast-as-possible OLAP system** and it still does not have full-fledged transaction support, but some features like consistent read/writes and mutations for updating/deleting data had to be added.

The fundamental trade-off between OLAP and OLTP systems remains:

- To build analytical reports efficiently it's crucial to be able to read columns separately, thus most OLAP databases are **columnar**,
- While storing columns separately increases costs of operations on rows, like append or in-place modification, proportionally to the number of columns (which can be huge if the systems try to collect all details of an event just in case). Thus, most OLTP systems store data arranged by rows.

## What Is a Columnar Database?

A columnar database stores data of each column independently. This allows to read data from disks only for those columns that are used in any given query. The cost is that operations that affect whole rows become proportionally more expensive. The synonym for a columnar database is a column-oriented database management system. ClickHouse is a typical example of such a system.

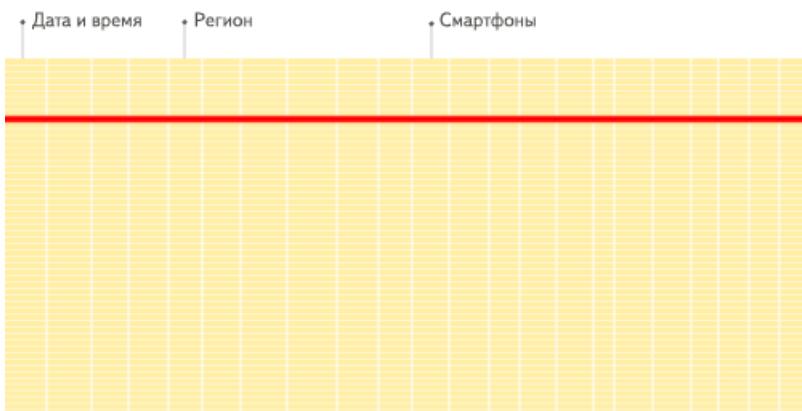
Key columnar database advantages are:

- Queries that use only a few columns out of many.
- Aggregating queries against large volumes of data.

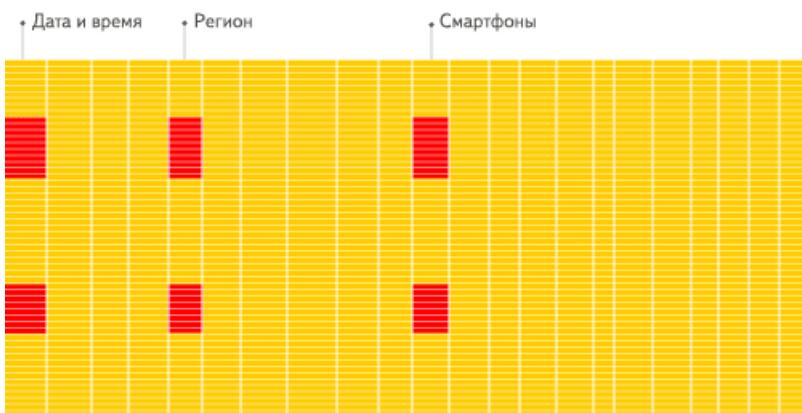
- Column-wise data compression.

Here is the illustration of the difference between traditional row-oriented systems and columnar databases when building reports:

### Traditional row-oriented



### Columnar



A columnar database is a preferred choice for analytical applications because it allows to have many columns in a table just in case, but do not pay the cost for unused columns on read query execution time. Column-oriented databases are designed for big data processing because and data warehousing, they often natively scale using distributed clusters of low-cost hardware to increase throughput. ClickHouse does it with combination of **distributed** and **replicated** tables.

## Why Not Use Something Like MapReduce?

We can refer to systems like MapReduce as distributed computing systems in which the reduce operation is based on distributed sorting. The most common open-source solution in this class is **Apache Hadoop**. Yandex uses its in-house solution, YT.

These systems aren't appropriate for online queries due to their high latency. In other words, they can't be used as the back-end for a web interface. These types of systems aren't useful for real-time data updates. Distributed sorting isn't the best way to perform reduce operations if the result of the operation and all the intermediate results (if there are any) are located in the RAM of a single server, which is usually the case for online queries. In such a case, a hash table is an optimal way to perform reduce operations. A common approach to optimizing map-reduce tasks is pre-aggregation (partial reduce) using a hash table in RAM. The user performs this optimization manually. Distributed sorting is one of the main causes of reduced performance when running simple map-reduce tasks.

Most MapReduce implementations allow you to execute arbitrary code on a cluster. But a declarative query language is better suited to OLAP to run experiments quickly. For example, Hadoop has Hive and Pig. Also consider Cloudera Impala or Shark (outdated) for Spark, as well as Spark SQL, Presto, and Apache

Drill. Performance when running such tasks is highly sub-optimal compared to specialized systems, but relatively high latency makes it unrealistic to use these systems as the backend for a web interface.

## Questions About ClickHouse Use Cases

Questions:

- [Can I use ClickHouse as a time-series database?](#)
- [Can I use ClickHouse as a key-value storage?](#)

### Don't see what you were looking for?

Check out [other F.A.Q. categories](#) or browse around main documentation articles found in the left sidebar.

## Can I Use ClickHouse As a Key-Value Storage?

The short answer is “**no**”. The key-value workload is among top positions in the list of cases when **NOT** to use ClickHouse. It's an [OLAP](#) system after all, while there are many excellent key-value storage systems out there.

However, there might be situations where it still makes sense to use ClickHouse for key-value-like queries. Usually, it's some low-budget products where the main workload is analytical in nature and fits ClickHouse well, but there's also some secondary process that needs a key-value pattern with not so high request throughput and without strict latency requirements. If you had an unlimited budget, you would have installed a secondary key-value database for thus secondary workload, but in reality, there's an additional cost of maintaining one more storage system (monitoring, backups, etc.) which might be desirable to avoid.

If you decide to go against recommendations and run some key-value-like queries against ClickHouse, here're some tips:

- The key reason why point queries are expensive in ClickHouse is its sparse primary index of main [MergeTree table engine family](#). This index can't point to each specific row of data, instead, it points to each N-th and the system has to scan from the neighboring N-th row to the desired one, reading excessive data along the way. In a key-value scenario, it might be useful to reduce the value of N with the [index\\_granularity](#) setting.
- ClickHouse keeps each column in a separate set of files, so to assemble one complete row it needs to go through each of those files. Their count increases linearly with the number of columns, so in the key-value scenario, it might be worth to avoid using many columns and put all your payload in a single [String](#) column encoded in some serialization format like JSON, Protobuf or whatever makes sense.
- There's an alternative approach that uses [Join](#) table engine instead of normal [MergeTree](#) tables and [joinGet](#) function to retrieve the data. It can provide better query performance but might have some usability and reliability issues. Here's an [usage example](#).

## Can I Use ClickHouse As a Time-Series Database?

ClickHouse is a generic data storage solution for **OLAP** workloads, while there are many specialized time-series database management systems. Nevertheless, ClickHouse's **focus on query execution speed** allows it to outperform specialized systems in many cases. There are many independent benchmarks on this topic out there, so we're not going to conduct one here. Instead, let's focus on ClickHouse features that are important to use if that's your use case.

First of all, there are **specialized codecs** which make typical time-series. Either common algorithms like `DoubleDelta` and `Gorilla` or specific to ClickHouse like `T64`.

Second, time-series queries often hit only recent data, like one day or one week old. It makes sense to use servers that have both fast nVME/SSD drives and high-capacity HDD drives. ClickHouse **TTL** feature allows to configure keeping fresh hot data on fast drives and gradually move it to slower drives as it ages. Rollup or removal of even older data is also possible if your requirements demand it.

Even though it's against ClickHouse philosophy of storing and processing raw data, you can use **materialized views** to fit into even tighter latency or costs requirements.

## Question About Operating ClickHouse Servers and Clusters

Questions:

- [Which ClickHouse version to use in production?](#)
- [Is it possible to delete old records from a ClickHouse table?](#)

### Don't see what you were looking for?

Check out [other F.A.Q. categories](#) or browse around main documentation articles found in the left sidebar.

## Which ClickHouse Version to Use in Production?

First of all, let's discuss why people ask this question in the first place. There are two key reasons:

1. ClickHouse is developed with pretty high velocity and usually, there are 10+ stable releases per year. It makes a wide range of releases to choose from, which is not so trivial choice.
2. Some users want to avoid spending time figuring out which version works best for their use case and just follow someone else's advice.

The second reason is more fundamental, so we'll start with it and then get back to navigating through various ClickHouse releases.

## Which ClickHouse Version Do You Recommend?

It's tempting to hire consultants or trust some known experts to get rid of responsibility for your production environment. You install some specific ClickHouse version that someone else recommended, now if there's some issue with it - it's not your fault, it's someone else's. This line of reasoning is a big trap. No external person knows better what's going on in your company's production environment.

So how to properly choose which ClickHouse version to upgrade to? Or how to choose your first ClickHouse version? First of all, you need to invest in setting up a **realistic pre-production environment**. In an ideal world, it could be a completely identical shadow copy, but that's usually expensive.

Here're some key points to get reasonable fidelity in a pre-production environment with not so high costs:

- Pre-production environment needs to run an as close set of queries as you intend to run in production:
  - Don't make it read-only with some frozen data.
  - Don't make it write-only with just copying data without building some typical reports.
  - Don't wipe it clean instead of applying schema migrations.
- Use a sample of real production data and queries. Try to choose a sample that's still representative and makes `SELECT` queries return reasonable results. Use obfuscation if your data is sensitive and internal policies do not allow it to leave the production environment.
- Make sure that pre-production is covered by your monitoring and alerting software the same way as your production environment does.
- If your production spans across multiple datacenters or regions, make your pre-production does the same.
- If your production uses complex features like replication, distributed table, cascading materialize views, make sure they are configured similarly in pre-production.
- There's a trade-off on using the roughly same number of servers or VMs in pre-production as in production, but of smaller size, or much less of them, but of the same size. The first option might catch extra network-related issues, while the latter is easier to manage.

The second area to invest in is **automated testing infrastructure**. Don't assume that if some kind of query has executed successfully once, it'll continue to do so forever. It's ok to have some unit tests where ClickHouse is mocked but make sure your product has a reasonable set of automated tests that are run against real ClickHouse and check that all important use cases are still working as expected.

Extra step forward could be contributing those automated tests to [ClickHouse's open-source test infrastructure](#) that's continuously used in its day-to-day development. It definitely will take some additional time and effort to learn [how to run it](#) and then how to adapt your tests to this framework, but it'll pay off by ensuring that ClickHouse releases are already tested against them when they are announced stable, instead of repeatedly losing time on reporting the issue after the fact and then waiting for a bugfix to be implemented, backported and released. Some companies even have such test contributions to infrastructure by its use as an internal policy, most notably it's called [Beyonce's Rule](#) at Google.

When you have your pre-production environment and testing infrastructure in place, choosing the best version is straightforward:

1. Routinely run your automated tests against new ClickHouse releases. You can do it even for ClickHouse releases that are marked as `testing`, but going forward to the next steps with them is not recommended.
2. Deploy the ClickHouse release that passed the tests to pre-production and check that all processes are running as expected.
3. Report any issues you discovered to [ClickHouse GitHub Issues](#).
4. If there were no major issues, it should be safe to start deploying ClickHouse release to your production environment. Investing in gradual release automation that implements an approach similar to [canary releases](#) or [green-blue deployments](#) might further reduce the risk of issues in production.

As you might have noticed, there's nothing specific to ClickHouse in the approach described above, people do that for any piece of infrastructure they rely on if they take their production environment seriously.

## How to Choose Between ClickHouse Releases?

If you look into contents of ClickHouse package repository, you'll see four kinds of packages:

1. testing
2. prestable
3. stable
4. Its (long-term support)

As was mentioned earlier, testing is good mostly to notice issues early, running them in production is not recommended because each of them is not tested as thoroughly as other kinds of packages.

prestable is a release candidate which generally looks promising and is likely to become announced as stable soon. You can try them out in pre-production and report issues if you see any.

For production use, there are two key options: stable and Its. Here is some guidance on how to choose between them:

- stable is the kind of package we recommend by default. They are released roughly monthly (and thus provide new features with reasonable delay) and three latest stable releases are supported in terms of diagnostics and backporting of bugfixes.
- Its are released twice a year and are supported for a year after their initial release. You might prefer them over stable in the following cases:
  - Your company has some internal policies that do not allow for frequent upgrades or using non-LTS software.
  - You are using ClickHouse in some secondary products that either does not require any complex ClickHouse features and do not have enough resources to keep it updated.

Many teams who initially thought that Its is the way to go, often switch to stable anyway because of some recent feature that's important for their product.

## Important

One more thing to keep in mind when upgrading ClickHouse: we're always keeping eye on compatibility across releases, but sometimes it's not reasonable to keep and some minor details might change. So make sure you check the [changelog](#) before upgrading to see if there are any notes about backward-incompatible changes.

# Is It Possible to Delete Old Records from a ClickHouse Table?

The short answer is “yes”. ClickHouse has multiple mechanisms that allow freeing up disk space by removing old data. Each mechanism is aimed for different scenarios.

## TTL

ClickHouse allows to automatically drop values when some condition happens. This condition is configured as an expression based on any columns, usually just static offset for any timestamp column.

The key advantage of this approach is that it does not need any external system to trigger, once TTL is configured, data removal happens automatically in background.

## Note

TTL can also be used to move data not only to [/dev/null](#), but also between different storage systems, like from SSD to HDD.

---

More details on [configuring TTL](#).

## ALTER DELETE

ClickHouse does not have real-time point deletes like in [OLTP](#) databases. The closest thing to them are mutations. They are issued as `ALTER ... DELETE` or `ALTER ... UPDATE` queries to distinguish from normal `DELETE` or `UPDATE` as they are asynchronous batch operations, not immediate modifications. The rest of syntax after `ALTER TABLE` prefix is similar.

`ALTER DELETE` can be issued to flexibly remove old data. If you need to do it regularly, the main downside will be the need to have an external system to submit the query. There are also some performance considerations since mutation rewrite complete parts even there's only a single row to be deleted.

This is the most common approach to make your system based on ClickHouse [GDPR](#)-compliant.

More details on [mutations](#).

## DROP PARTITION

`ALTER TABLE ... DROP PARTITION` provides a cost-efficient way to drop a whole partition. It's not that flexible and needs proper partitioning scheme configured on table creation, but still covers most common cases. Like mutations need to be executed from an external system for regular use.

More details on [manipulating partitions](#).

## TRUNCATE

It's rather radical to drop all data from a table, but in some cases it might be exactly what you need.

More details on [table truncation](#).

---

# Questions About Integrating ClickHouse and Other Systems

Questions:

- [How do I export data from ClickHouse to a file?](#)
- [How to import JSON into ClickHouse?](#)
- [What if I have a problem with encodings when connecting to Oracle via ODBC?](#)

**Don't see what you were looking for?**

Check out [other F.A.Q. categories](#) or browse around main documentation articles found in the left sidebar.

---

# How Do I Export Data from ClickHouse to a File? Using INTO OUTFILE Clause

Add an `INTO OUTFILE` clause to your query.

For example:

```
SELECT * FROM table INTO OUTFILE 'file'
```

By default, ClickHouse uses the [TabSeparated](#) format for output data. To select the [data format](#), use the [FORMAT clause](#).

For example:

```
SELECT * FROM table INTO OUTFILE 'file' FORMAT CSV
```

## Using a File-Engine Table

See [File](#) table engine.

## Using Command-Line Redirection

```
$ clickhouse-client --query "SELECT * from table" --format FormatName > result.txt
```

See [clickhouse-client](#).

## How to Import JSON Into ClickHouse?

ClickHouse supports a wide range of [data formats for input and output](#). There are multiple JSON variations among them, but the most commonly used for data ingestion is [JSONEachRow](#). It expects one JSON object per row, each object separated by a newline.

## Examples

Using [HTTP interface](#):

```
$ echo '{"foo":"bar"}' | curl 'http://localhost:8123/?query=INSERT%20INTO%20test%20FORMAT%20JSONEachRow' --data-binary @-
```

Using [CLI interface](#):

```
$ echo '{"foo":"bar"}' | clickhouse-client --query="INSERT INTO test FORMAT JSONEachRow"
```

Instead of inserting data manually, you might consider to use one of [client libraries](#) instead.

## Useful Settings

- `input_format_skip_unknown_fields` allows to insert JSON even if there were additional fields not present in table schema (by discarding them).
- `input_format_import_nested_json` allows to insert nested JSON objects into columns of [Nested](#) type.

### Note

Settings are specified as [GET](#) parameters for the HTTP interface or as additional command-line arguments prefixed with `--` for the [CLI](#) interface.

# What If I Have a Problem with Encodings When Using Oracle Via ODBC?

If you use Oracle as a source of ClickHouse external dictionaries via Oracle ODBC driver, you need to set the correct value for the `NLS_LANG` environment variable in `/etc/default/clickhouse`. For more information, see the [Oracle NLS\\_LANG FAQ](#).

## Example

```
NLS_LANG=RUSSIAN_RUSSIA.UTF8
```

# ClickHouse F.A.Q

This section of the documentation is a place to collect answers to ClickHouse-related questions that arise often.

Categories:

- **General**

- [What is ClickHouse?](#)
- [Why ClickHouse is so fast?](#)
- [Who is using ClickHouse?](#)
- [What does “ClickHouse” mean?](#)
- [What does “Не тормозит” mean?](#)
- [What is OLAP?](#)
- [What is a columnar database?](#)
- [Why not use something like MapReduce?](#)

- **Use Cases**

- [Can I use ClickHouse as a time-series database?](#)
- [Can I use ClickHouse as a key-value storage?](#)

- **Operations**

- [Which ClickHouse version to use in production?](#)
- [Is it possible to delete old records from a ClickHouse table?](#)

- **Integration**

- [How do I export data from ClickHouse to a file?](#)
- [What if I have a problem with encodings when connecting to Oracle via ODBC?](#)

ClickHouse release v21.10, 2021-10-16

Backward Incompatible Change

- Now the following MergeTree table-level settings: `replicated_max_parallel_sends`, `replicated_max_parallel_sends_for_table`, `replicated_max_parallel_fetches`, `replicated_max_parallel_fetches_for_table` do nothing. They never worked well and were replaced with `max_replicated_fetches_network_bandwidth`, `max_replicated_sends_network_bandwidth` and `background_fetches_pool_size`. #28404 (alesapin).

## New Feature

- Add feature for creating user-defined functions (UDF) as lambda expressions. Syntax `CREATE FUNCTION {function_name} as ({parameters}) -> {function core}`. Example `CREATE FUNCTION plus_one as (a) -> a + 1`. Authors @Realist007. #27796 (Maksim Kita) #23978 (Realist007).
- Added Executable storage engine and `executable` table function. It enables data processing with external scripts in streaming fashion. #28102 (Maksim Kita) (ruct).
- Added ExecutablePool storage engine. Similar to `Executable` but it's using a pool of long running processes. #28518 (Maksim Kita).
- Add `ALTER TABLE ... MATERIALIZE COLUMN` query. #27038 (Vladimir Chebotarev).
- Support for partitioned write into `s3` table function. #23051 (Vladimir Chebotarev).
- Support `lz4` compression format (in addition to `gz`, `bz2`, `xz`, `zstd`) for data import / export. #25310 (Bharat Nallan).
- Allow positional arguments under setting `enable_positional_arguments`. Closes #2592. #27530 (Ksenia Sumarokova).
- Accept user settings related to file formats in `SETTINGS` clause in `CREATE` query for `s3` tables. This closes #27580. #28037 (Nikita Mikhaylov).
- Allow SSL connection for `RabbitMQ` engine. #28365 (Ksenia Sumarokova).
- Add `getServerPort` function to allow getting server port. When the port is not used by the server, throw an exception. #27900 (Amos Bird).
- Add conversion functions between "snowflake id" and `DateTime`, `DateTime64`. See #27058. #27704 (jasine).
- Add function `SHA512`. #27830 (zhanglistar).
- Add `log_queries_probability` setting that allows user to write to `query_log` only a sample of queries. Closes #16609. #27527 (Nikolay Degterinsky).

## Experimental Feature

- `web` type of disks to store readonly tables on web server in form of static files. See #23982. #25251 (Ksenia Sumarokova). This is mostly needed to facilitate testing of operation on shared storage and for easy importing of datasets. Not recommended to use before release 21.11.
- Added new commands `BACKUP` and `RESTORE`. #21945 (Vitaly Baranov). This is under development and not intended to be used in current version.

## Performance Improvement

- Speed up `sumIf` and `countIf` aggregation functions. #28272 (Raúl Marín).
- Create virtual projection for `minmax` indices. Now, when `allow_experimental_projection_optimization` is enabled, queries will use minmax index instead of reading the data when possible. #26286 (Amos Bird).

- Introducing two checks in `sequenceMatch` and `sequenceCount` that allow for early exit when some deterministic part of the sequence pattern is missing from the events list. This change unlocks many queries that would previously fail due to reaching operations cap, and generally speeds up the pipeline. [#27729 \(Jakub Kuklis\)](#).
- Enhance primary key analysis with always monotonic information of binary functions, notably non-zero constant division. [#28302 \(Amos Bird\)](#).
- Make `hasAll` filter condition leverage bloom filter data-skipping indexes. [#27984 \(Braulio Valdivielso Martínez\)](#).
- Speed up data parts loading by delaying table startup process. [#28313 \(Amos Bird\)](#).
- Fixed possible excessive number of conditions moved from `WHERE` to `PREWHERE` (optimization controlled by settings `optimize_move_to_prewhere`). [#28139 \(lthaooo\)](#).
- Enable `optimize_distributed_group_by_sharding_key` by default. [#28105 \(Azat Khuzhin\)](#).

## Improvement

- Check cluster name before creating `Distributed` table, do not allow to create a table with incorrect cluster name. Fixes [#27832](#). [#27927 \(tavplubix\)](#).
- Add aggregate function `quantileBFLOAT16Weighted` similarly to other quantile...Weighted functions. This closes [#27745](#). [#27758 \(Ivan Novitskiy\)](#).
- Allow to create dictionaries with empty attributes list. [#27905 \(Maksim Kita\)](#).
- Add interactive documentation in `clickhouse-client` about how to reset the password. This is useful in scenario when user has installed ClickHouse, set up the password and instantly forget it. See [#27750](#). [#27903 \(alexey-milovidov\)](#).
- Support the case when the data is enclosed in array in `JSONAsString` input format. Closes [#25517](#). [#25633 \(Kruglov Pavel\)](#).
- Add new column `last_queue_update_exception` to `system.replicas` table. [#26843 \(nvartolomei\)](#).
- Support reconnections on failover for `MaterializedPostgreSQL` tables. Closes [#28529](#). [#28614 \(Kseniia Sumarokova\)](#).
- Generate a unique server UUID on first server start. [#20089 \(Bharat Nallan\)](#).
- Introduce `connection_wait_timeout` (default to 5 seconds, 0 - do not wait) setting for `MySQL` engine. [#28474 \(Azat Khuzhin\)](#).
- Do not allow creating `MaterializedPostgreSQL` with bad arguments. Closes [#28423](#). [#28430 \(Kseniia Sumarokova\)](#).
- Use real tmp file instead of predefined "rows\_sources" for vertical merges. This avoids generating garbage directories in tmp disks. [#28299 \(Amos Bird\)](#).
- Added `libhdfs3_conf` in server config instead of export env `LIBHDFS3_CONF` in `clickhouse-server.service`. This is for configuration of interaction with HDFS. [#28268 \(Zhichang Yu\)](#).
- Fix removing of parts in a Temporary state which can lead to an unexpected exception (`Part %name% doesn't exist`). Fixes [#23661](#). [#28221 #28221 \(Azat Khuzhin\)](#).
- Fix `zookeeper_log.address` (before the first patch in this PR the address was always ::) and reduce number of calls `getpeername(2)` for this column (since each time entry for `zookeeper_log` is added `getpeername()` is called, cache this address in the zookeeper client to avoid this). [#28212 \(Azat Khuzhin\)](#).

- Support implicit conversions between index in operator [] and key of type `Map` (e.g. different `Int` types, `String` and `FixedString`). #28096 (Anton Popov).
- Support `ON CONFLICT` clause when inserting into PostgreSQL table engine or table function. Closes #27727. #28081 (Ksenia Sumarokova).
- Lower restrictions for `Enum` data type to allow attaching compatible data. Closes #26672. #28028 (Dmitry Novik).
- Add a setting `empty_result_for_aggregation_by_constant_keys_on_empty_set` to control the behavior of grouping by constant keys on empty set. This is to bring back the old behaviour of #6842. #27932 (Amos Bird).
- Added `replication_wait_for_inactive_replica_timeout` setting. It allows to specify how long to wait for inactive replicas to execute `ALTER/OPTIMZE/TRUNCATE` query (default is 120 seconds). If `replication_alter_partitions_sync` is 2 and some replicas are not active for more than `replication_wait_for_inactive_replica_timeout` seconds, then `UNFINISHED` will be thrown. #27931 (tavplubix).
- Support lambda argument for `APPLY` column transformer which allows applying functions with more than one argument. This is for #27877. #27901 (Amos Bird).
- Enable `tcp_keep_alive_timeout` by default. #27882 (Azat Khuzhin).
- Improve remote query cancelation (in case of remote server abnormaly terminated). #27881 (Azat Khuzhin).
- Use Multipart copy upload for large S3 objects. #27858 (ianton-ru).
- Allow symlink traversal for library dictionary path. #27815 (Ksenia Sumarokova).
- Now `ALTER MODIFY COLUMN T` to `Nullable(T)` doesn't require mutation. #27787 (victorgao).
- Don't silently ignore errors and don't count delays in `ReadBufferFromS3`. #27484 (Vladimir Chebotarev).
- Improve `ALTER ... MATERIALIZE TTL` by recalculating metadata only without actual TTL action. #27019 (lthaooo).
- Allow reading the list of custom top level domains without a new line at EOF. #28213 (Azat Khuzhin).

## Bug Fix

- Fix cases, when reading compressed data from carbon-clickhouse fails with 'attempt to read after end of file'. Closes #26149. #28150 (FArthur-cmd).
- Fix checking access grants when executing `GRANT WITH REPLACE` statement with `ON CLUSTER` clause. This PR improves fix #27001. #27983 (Vitaly Baranov).
- Allow selecting with `extremes = 1` from a column of the type `LowCardinality(UUID)`. #27918 (Vitaly Baranov).
- Fix PostgreSQL-style cast (`::` operator) with negative numbers. #27876 (Anton Popov).
- After #26864. Fix shutdown of `NamedSessionStorage`: session contexts stored in `NamedSessionStorage` are now destroyed before destroying the global context. #27875 (Vitaly Baranov).
- Bugfix for `windowFunnel` "strict" mode. This fixes #27469. #27563 (achimbab).
- Fix infinite loop while reading truncated `bzip2` archive. #28543 (Azat Khuzhin).
- Fix UUID overlap in `DROP TABLE` for internal DDL from `MaterializedMySQL`. `MaterializedMySQL` is an experimental feature. #28533 (Azat Khuzhin).

- Fix There is no subcolumn error, while select from tables, which have Nested columns and scalar columns with dot in name and the same prefix as Nested (e.g. n.id UInt32, n.arr1 Array(UInt64), n.arr2 Array(UInt64)). #28531 (Anton Popov).
- Fix bug which can lead to error Existing table metadata in ZooKeeper differs in sorting key expression.after ALTER of ReplicatedVersionedCollapsingMergeTree. Fixes #28515. #28528 (alesapin).
- Fixed possible ZooKeeper watches leak (minor issue) on background processing of distributed DDL queue. Closes #26036. #28446 (tavplubix).
- Fix missing quoting of table names in MaterializedPostgreSQL engine. Closes #28316. #28433 (Kseniia Sumarokova).
- Fix the wrong behaviour of non joined rows from nullable column. Close #27691. #28349 (vdimir).
- Fix NOT-IN index optimization when not all key columns are used. This fixes #28120. #28315 (Amos Bird).
- Fix intersecting parts due to new part had been replaced with an empty part. #28310 (Azat Khuzhin).
- Fix inconsistent result in queries with ORDER BY and Merge tables with enabled setting optimize\_read\_in\_order. #28266 (Anton Popov).
- Fix possible read of uninitialized memory for queries with Nullable(LowCardinality) type and the setting extremes set to 1. Fixes #28165. #28205 (Nikolai Kochetov).
- Multiple small fixes for projections. See detailed description in the PR. #28178 (Amos Bird).
- Fix extremely rare segfaults on shutdown due to incorrect order of context/config reloader shutdown. #28088 (nvartolomei).
- Fix handling null value with type of Nullable(String) in function JSONExtract. This fixes #27929 and #27930. This was introduced in <https://github.com/ClickHouse/ClickHouse/pull/25452>. #27939 (Amos Bird).
- Multiple fixes for the new clickhouse-keeper tool. Fix a rare bug in clickhouse-keeper when the client can receive a watch response before request-response. #28197 (alesapin). Fix incorrect behavior in clickhouse-keeper when list watches (getChildren) triggered with set requests for children. #28190 (alesapin). Fix rare case when changes of clickhouse-keeper settings may lead to lost logs and server hung. #28360 (alesapin). Fix bug in clickhouse-keeper which can lead to endless logs when rotate\_logs\_interval decreased. #28152 (alesapin).

## Build/Testing/Packaging Improvement

- Enable Thread Fuzzer in Stress Test. Thread Fuzzer is ClickHouse feature that allows to test more permutations of thread scheduling and discover more potential issues. This closes #9813. This closes #9814. This closes #9515. This closes #9516. #27538 (alexey-milovidov).
- Add new log level test for testing environments. It is even more verbose than the default trace. #28559 (alesapin).
- Print out git status information at CMake configure stage. #28047 (Braulio Valdivielso Martínez).
- Temporarily switched ubuntu apt repository to mirror ru.archive.ubuntu.com as the default one (archive.ubuntu.com) is not responding from our CI. #28016 (Ilya Yatsishin).

# ClickHouse release v21.9, 2021-09-09

## Backward Incompatible Change

- Do not output trailing zeros in text representation of `Decimal` types. Example: `1.23` will be printed instead of `1.230000` for decimal with scale 6. This closes [#15794](#). It may introduce slight incompatibility if your applications somehow relied on the trailing zeros. Serialization in output formats can be controlled with the setting `output_format_decimal_trailing_zeros`. Implementation of `toString` and casting to `String` is changed unconditionally. [#27680](#) ([alexey-milovidov](#)).
- Do not allow to apply parametric aggregate function with `-Merge` combinator to aggregate function state if state was produced by aggregate function with different parameters. For example, state of `fooState(42)(x)` cannot be finalized with `fooMerge(s)` or `fooMerge(123)(s)`, parameters must be specified explicitly like `fooMerge(42)(s)` and must be equal. It does not affect some special aggregate functions like `quantile` and `sequence*` that use parameters for finalization only. [#26847](#) ([tavplubix](#)).
- Under `clickhouse-local`, always treat local addresses with a port as remote. [#26736](#) ([Raúl Marín](#)).
- Fix the issue that in case of some sophisticated query with column aliases identical to the names of expressions, bad cast may happen. This fixes [#25447](#). This fixes [#26914](#). This fix may introduce backward incompatibility: if there are different expressions with identical names, exception will be thrown. It may break some rare cases when `enable_optimize_predicate_expression` is set. [#26639](#) ([alexey-milovidov](#)).
- Now, scalar subquery always returns `Nullable` result if it's type can be `Nullable`. It is needed because in case of empty subquery it's result should be `Null`. Previously, it was possible to get error about incompatible types (type deduction does not execute scalar subquery, and it could use not-nullable type). Scalar subquery with empty result which can't be converted to `Nullable` (like `Array` or `Tuple`) now throws error. Fixes [#25411](#). [#26423](#) ([Nikolai Kochetov](#)).
- Introduce syntax for here documents. Example `SELECT $doc$ VALUE $doc$`. [#26671](#) ([Maksim Kita](#)). This change is backward incompatible if in query there are identifiers that contain `$`. [#28768](#).
- Now indices can handle `Nullable` types, including `isNull` and `isNotNull`. [#12433](#) and [#12455](#) ([Amos Bird](#)) and [#27250](#) ([Azat Khuzhin](#)). But this was done with on-disk format changes, and even though new server can read old data, old server cannot. Also, in case you have `MINMAX` data skipping indices, you may get `Data after mutation/merge is not byte-identical` error, since new index will have `.idx2` extension while before it was `.idx`. That said, that you should not delay updating all existing replicas, in this case, otherwise, if old replica (<21.9) will download data from new replica with 21.9+ it will not be able to apply index for downloaded part.

## New Feature

- Implementation of short circuit function evaluation, closes [#12587](#). Add settings `short_circuit_function_evaluation` to configure short circuit function evaluation. [#23367](#) ([Kruglov Pavel](#)).
- Add support for `INTERSECT`, `EXCEPT`, `ANY`, `ALL` operators. [#24757](#) ([Kirill Ershov](#)). ([Kseniia Sumarokova](#)).
- Add support for encryption at the virtual file system level (data encryption at rest) using AES-CTR algorithm. [#24206](#) ([Latysheva Alexandra](#)). ([Vitaly Baranov](#)) [#26733](#) [#26377](#) [#26465](#).
- Added natural language processing (NLP) functions for tokenization, stemming, lemmatizing and search in synonyms extensions. [#24997](#) ([Nikolay Degterinsky](#)).
- Added integration with S2 geometry library. [#24980](#) ([Andr0901](#)). ([Nikita Mikhaylov](#)).
- Add SQLite table engine, table function, database engine. [#24194](#) ([Arslan Gumerov](#)). ([Kseniia Sumarokova](#)).
- Added support for custom query for MySQL, PostgreSQL, ClickHouse, JDBC, Cassandra dictionary source. Closes [#1270](#). [#26995](#) ([Maksim Kita](#)).

- Add shared (replicated) storage of user, roles, row policies, quotas and settings profiles through ZooKeeper. #27426 (Kevin Michel).
- Add compression for `INTO OUTFILE` that automatically choose compression algorithm. Closes #3473. #27134 (Filatenkov Artur).
- Add `INSERT ... FROM INFILE` similarly to `SELECT ... INTO OUTFILE`. #27655 (Filatenkov Artur).
- Added `complex_key_range_hashed` dictionary. Closes #22029. #27629 (Maksim Kita).
- Support expressions in `JOIN ON` section. Close #21868. #24420 (Vladimir C).
- When client connects to server, it receives information about all warnings that are already were collected by server. (It can be disabled by using option `--no-warnings`). Add `system.warnings` table to collect warnings about server configuration. #26246 (Filatenkov Artur). #26282 (Filatenkov Artur).
- Allow using constant expressions from with and select in aggregate function parameters. Close #10945. #27531 (abel-cheng).
- Add `tupleToNameValuePairs`, a function that turns a named tuple into an array of pairs. #27505 (Braulio Valdivielso Martínez).
- Add support for `bzip2` compression method for import/export. Closes #22428. #27377 (Nikolay Degterinsky).
- Added `bitmapSubsetOffsetLimit(bitmap, offset, cardinality_limit)` function. It creates a subset of bitmap limit the results to `cardinality_limit` with offset of `offset`. #27234 (DHBin).
- Add column `default_database` to `system.users`. #27054 (kevin wan).
- Supported `cluster` macros inside table functions 'cluster' and 'clusterAllReplicas'. #26913 (polyprogrammist).
- Add new functions `currentRoles()`, `enabledRoles()`, `defaultRoles()`. #26780 (Vitaly Baranov).
- New functions `currentProfiles()`, `enabledProfiles()`, `defaultProfiles()`. #26714 (Vitaly Baranov).
- Add functions that return `(initial_)query_id` of the current query. This closes #23682. #26410 (Alexey Boykov).
- Add `REPLACE GRANT` feature. #26384 (Caspian).
- `EXPLAIN` query now has `EXPLAIN ESTIMATE ...` mode that will show information about read rows, marks and parts from MergeTree tables. Closes #23941. #26131 (fastio).
- Added `system.zookeeper_log` table. All actions of ZooKeeper client are logged into this table. Implements #25449. #26129 (tavplubix).
- Zero-copy replication for `ReplicatedMergeTree` over `HDFS` storage. #25918 (Zhichang Yu).
- Allow to insert Nested type as array of structs in `Arrow`, `ORC` and `Parquet` input format. #25902 (Kruglov Pavel).
- Add a new datatype `Date32` (store data as `Int32`), support date range same with `DateTime64` support load parquet date32 to ClickHouse `Date32` Add new function `toDate32` like `toDate`. #25774 (LiuNeng).
- Allow setting default database for users. #25268. #25687 (kevin wan).
- Add an optional parameter to `MongoDB` engine to accept connection string options and support SSL connection. Closes #21189. Closes #21041. #22045 (Omar Bazaraa).

## Experimental Feature

- Added a compression codec `AES_128_GCM_SIV` which encrypts columns instead of compressing them. [#19896 \(PHO\)](#). Will be rewritten, do not use.
- Rename `MaterializeMySQL` to `MaterializedMySQL`. [#26822 \(tavplubix\)](#).

## Performance Improvement

- Improve the performance of fast queries when `max_execution_time = 0` by reducing the number of `clock_gettime` system calls. [#27325 \(filimonov\)](#).
- Specialize date time related comparison to achieve better performance. This fixes [#27083](#) . [#27122 \(Amos Bird\)](#).
- Share file descriptors in concurrent reads of the same files. There is no noticeable performance difference on Linux. But the number of opened files will be significantly (10..100 times) lower on typical servers and it makes operations easier. See [#26214](#). [#26768 \(alexey-milovidov\)](#).
- Improve latency of short queries, that require reading from tables with large number of columns. [#26371 \(Anton Popov\)](#).
- Don't build sets for indices when analyzing a query. [#26365 \(Raúl Marín\)](#).
- Vectorize the SUM of Nullable integer types with native representation ([David Manzanares](#), [Raúl Marín](#)). [#26248 \(Raúl Marín\)](#).
- Compile expressions involving columns with `Enum` types. [#26237 \(Maksim Kita\)](#).
- Compile aggregate functions `groupBitOr`, `groupBitAnd`, `groupBitXor`. [#26161 \(Maksim Kita\)](#).
- Improved memory usage with better block size prediction when reading empty DEFAULT columns. Closes [#17317](#). [#25917 \(Vladimir Chebotarev\)](#).
- Reduce memory usage and number of read rows in queries with `ORDER BY primary_key`. [#25721 \(Anton Popov\)](#).
- Enable `distributed_push_down_limit` by default. [#27104 \(Azat Khuzhin\)](#).
- Make `toTimeZone` monotonicity when `timeZone` is a constant value to support partition purging when use sql like:. [#26261 \(huangzhaowei\)](#).

## Improvement

- Mark window functions as ready for general use. Remove the `allow_experimental_window_functions` setting. [#27184 \(Alexander Kuzmenkov\)](#).
- Improve compatibility with non-whole-minute timezone offsets. [#27080 \(Raúl Marín\)](#).
- If file descriptor in `File` table is regular file - allow to read multiple times from it. It allows `clickhouse-local` to read multiple times from `stdin` (with multiple `SELECT` queries or subqueries) if `stdin` is a regular file like `clickhouse-local --query "SELECT * FROM table UNION ALL SELECT * FROM table" ... < file` This closes [#11124](#). Co-authored with ([alexey-milovidov](#)). [#25960 \(BoloniniD\)](#).
- Remove duplicate index analysis and avoid possible invalid limit checks during projection analysis. [#27742 \(Amos Bird\)](#).
- Enable query parameters to be passed in the body of HTTP requests. [#27706 \(Hermano Lustosa\)](#).
- Disallow `arrayJoin` on partition expressions. [#27648 \(Raúl Marín\)](#).
- Log client IP address if authentication fails. [#27514 \(Misko Lee\)](#).

- Use bytes instead of strings for binary data in the GRPC protocol. #27431 (Vitaly Baranov).
- Send response with error message if HTTP port is not set and user tries to send HTTP request to TCP port. #27385 (Braulio Valdivielso Martínez).
- Add `_CAST` function for internal usage, which will not preserve type nullability, but non-internal cast will preserve according to setting `cast_keep_nullable`. Closes #12636. #27382 (Kseniia Sumarokova).
- Add setting `log_formatted_queries` to log additional formatted query into `system.query_log`. It's useful for normalized query analysis because functions like `normalizeQuery` and `normalizeQueryKeepNames` don't parse/format queries in order to achieve better performance. #27380 (Amos Bird).
- Add two settings `max_hyperscan_regexp_length` and `max_hyperscan_regexp_total_length` to prevent huge regexp being used in hyperscan related functions, such as `multiMatchAny`. #27378 (Amos Bird).
- Memory consumed by bitmap aggregate functions now is taken into account for memory limits. This closes #26555. #27252 (alexey-milovidov).
- Add 10 seconds cache for S3 proxy resolver. #27216 (ianton-ru).
- Split global mutex into individual regexp construction. This helps avoid huge regexp construction blocking other related threads. #27211 (Amos Bird).
- Support schema for PostgreSQL database engine. Closes #27166. #27198 (Kseniia Sumarokova).
- Track memory usage in clickhouse-client. #27191 (Filatenkov Artur).
- Try recording `query_kind` in `system.query_log` even when query fails to start. #27182 (Amos Bird).
- Added columns `replica_is_active` that maps replica name to is replica active status to table `system.replicas`. Closes #27138. #27180 (Maksim Kita).
- Allow to pass query settings via server URI in Web UI. #27177 (kolsys).
- Add a new metric called `MaxPushedDDLEntryID` which is the maximum ddl entry id that current node push to zookeeper. #27174 (Fuwang Hu).
- Improved the existence condition judgment and empty string node judgment when `clickhouse-keeper` creates znode. #27125 (小路).
- Merge JOIN correctly handles empty set in the right. #27078 (Vladimir C).
- Now functions can be shard-level constants, which means if it's executed in the context of some distributed table, it generates a normal column, otherwise it produces a constant value. Notable functions are: `hostName()`, `tcpPort()`, `version()`, `buildId()`, `uptime()`, etc. #27020 (Amos Bird).
- Updated `extractAllGroupsHorizontal` - upper limit on the number of matches per row can be set via optional third argument. #26961 (Vasily Nemkov).
- Expose RocksDB statistics via `system.rocksdb` table. Read rocksdb options from ClickHouse config (`rocksdb...` keys). NOTE: ClickHouse does not rely on RocksDB, it is just one of the additional integration storage engines. #26821 (Azat Khuzhin).
- Less verbose internal RocksDB logs. NOTE: ClickHouse does not rely on RocksDB, it is just one of the additional integration storage engines. This closes #26252. #26789 (alexey-milovidov).
- Changing default roles affects new sessions only. #26759 (Vitaly Baranov).
- Watchdog is disabled in docker by default. Fix for not handling `ctrl+c`. #26757 (Mikhail f. Shiryaev).
- SET PROFILE now applies constraints too if they're set for a passed profile. #26730 (Vitaly Baranov).

- Improve handling of `KILL QUERY` requests. [#26675](#) ([Raúl Marín](#)).
- `mapPopulatesSeries` function supports `Map` type. [#26663](#) ([Ildus Kurbangaliev](#)).
- Fix excessive (x2) connect attempts with `skip_unavailable_shards`. [#26658](#) ([Azat Khuzhin](#)).
- Avoid hanging `clickhouse-benchmark` if connection fails (i.e. on `EMFILE`). [#26656](#) ([Azat Khuzhin](#)).
- Allow more threads to be used by the Kafka engine. [#26642](#) ([feihengye](#)).
- Add round-robin support for `clickhouse-benchmark` (it does not differ from the regular multi host/port run except for statistics report). [#26607](#) ([Azat Khuzhin](#)).
- Executable dictionaries (`executable`, `executable_pool`) enable creation with DDL query using `clickhouse-local`. Closes [#22355](#). [#26510](#) ([Maksim Kita](#)).
- Set client query kind for `mysql` and `postgresql` compatibility protocol handlers. [#26498](#) ([anneji-dev](#)).
- Apply `LIMIT` on the shards for queries like `SELECT * FROM dist ORDER BY key LIMIT 10` w/  
`distributed_push_down_limit=1`. Avoid running `Distinct/LIMIT BY` steps for queries like `SELECT DISTINCT shading_key FROM dist ORDER BY key`. Now `distributed_push_down_limit` is respected by `optimize_distributed_group_by_sharding_key` optimization. [#26466](#) ([Azat Khuzhin](#)).
- Updated protobuf to 3.17.3. Changelogs are available on  
<https://github.com/protocolbuffers/protobuf/releases>. [#26424](#) ([Ilya Yatsishin](#)).
- Enable `use_hedged_requests` setting that allows to mitigate tail latencies on large clusters. [#26380](#) ([alexey-milovidov](#)).
- Improve behaviour with non-existing host in user allowed host list. [#26368](#) ([ianton-ru](#)).
- Add ability to set `Distributed` directory monitor settings via `CREATE TABLE` (i.e. `CREATE TABLE dist (key Int Engine=Distributed(cluster, db, table) SETTINGS monitor_batch_inserts=1` and similar). [#26336](#) ([Azat Khuzhin](#)).
- Save server address in history URLs in web UI if it differs from the origin of web UI. This closes [#26044](#).  
[#26322](#) ([alexey-milovidov](#)).
- Add events to profile calls to `sleep` / `sleepEachRow`. [#26320](#) ([Raúl Marín](#)).
- Allow to reuse connections of shards among different clusters. It also avoids creating new connections when using `cluster` table function. [#26318](#) ([Amos Bird](#)).
- Control the execution period of clear old temporary directories by parameter with default value.  
[#26212](#). [#26313](#) ([fastio](#)).
- Add a setting `function_range_max_elements_in_block` to tune the safety threshold for data volume generated by function `range`. This closes [#26303](#). [#26305](#) ([alexey-milovidov](#)).
- Check hash function at table creation, not at sampling. Add settings for MergeTree, if someone create a table with incorrect sampling column but sampling never be used, disable this settings for starting the server without exception. [#26256](#) ([zhaoyu](#)).
- Added `output_format_avro_string_column_pattern` setting to put specified String columns to Avro as string instead of default bytes. Implements [#22414](#). [#26245](#) ([Ilya Golshtein](#)).
- Add information about column sizes in `system.columns` table for `Log` and `TinyLog` tables. This closes [#9001](#).  
[#26241](#) ([Nikolay Degterinsky](#)).
- Don't throw exception when querying `system.detached_parts` table if there is custom disk configuration and `detached` directory does not exist on some disks. This closes [#26078](#). [#26236](#) ([alexey-milovidov](#)).

- Check for non-deterministic functions in keys, including constant expressions like `now()`, `today()`. This closes #25875. This closes #11333. #26235 (alexey-milovidov).
- convert timestamp and timestamptz data types to `DateTime64` in PostgreSQL table engine. #26234 (jasine).
- Apply aggressive IN index analysis for projections so that better projection candidate can be selected. #26218 (Amos Bird).
- Remove GLOBAL keyword for IN when scalar function is passed. In previous versions, if user specified `GLOBAL IN f(x)` exception was thrown. #26217 (Amos Bird).
- Add error id (like `BAD_ARGUMENTS`) to exception messages. This closes #25862. #26172 (alexey-milovidov).
- Fix incorrect output with --progress option for clickhouse-local. Progress bar will be cleared once it gets to 100% - same as it is done for clickhouse-client. Closes #17484. #26128 (Ksenia Sumarokova).
- Add `merge_selecting_sleep_ms` setting. #26120 (lthaooo).
- Remove complicated usage of Linux AIO with one block readahead and replace it with plain simple synchronous IO with `O_DIRECT`. In previous versions, the setting `min_bytes_to_use_direct_io` may not work correctly if `max_threads` is greater than one. Reading with direct IO (that is disabled by default for queries and enabled by default for large merges) will work in less efficient way. This closes #25997. #26003 (alexey-milovidov).
- Flush Distributed table on `REPLACE TABLE` query. Resolves #24566 - Do not replace (or create) table on `[CREATE OR] REPLACE TABLE ... AS SELECT` query if insertion into new table fails. Resolves #23175. #25895 (tavplubix).
- Add `views` column to `system.query_log` containing the names of the (materialized or live) views executed by the query. Adds a new log table (`system.query_views_log`) that contains information about each view executed during a query. Modifies view execution: When an exception is thrown while executing a view, any view that has already started will continue running until it finishes. This used to be the behaviour under `parallel_view_processing=true` and now it's always the same behaviour. - Dependent views now report reading progress to the context. #25714 (Raúl Marín).
- Do connection draining asynchronously upon finishing executing distributed queries. A new server setting is added `max_threads_for_connection_collector` which specifies the number of workers to recycle connections in background. If the pool is full, connection will be drained synchronously but a bit different than before: It's drained after we send EOS to client, query will succeed immediately after receiving enough data, and any exception will be logged instead of throwing to the client. Added setting `drain_timeout` (3 seconds by default). Connection draining will disconnect upon timeout. #25674 (Amos Bird).
- Support for multiple includes in configuration. It is possible to include users configuration, remote servers configuration from multiple sources. Simply place `<include />` element with `from_zk`, `from_env` or `incl` attribute and it will be replaced with the substitution. #24404 (nvartolomei).
- Fix multiple block insertion into distributed table with `insert_distributed_one_random_shard = 1`. This is a marginal feature. Mark as improvement. #23140 (Amos Bird).
- Support `LowCardinality` and `FixedString` keys/values for `Map` type. #21543 (hexiaoting).
- Enable reloading of local disk config. #19526 (taiyang-li).

## Bug Fix

- Fix a couple of bugs that may cause replicas to diverge. #27808 (tavplubix).

- Fix a rare bug in `DROP PART` which can lead to the error `Unexpected merged part intersects drop range`. [#27807 \(alesapin\)](#).
- Prevent crashes for some formats when `NULL` (tombstone) message was coming from Kafka. Closes [#19255](#). [#27794 \(filimonov\)](#).
- Fix column filtering with union distinct in subquery. Closes [#27578](#). [#27689 \(Kseniia Sumarokova\)](#).
- Fix bad type cast when functions like `arrayHas` are applied to arrays of `LowCardinality` of `Nullable` of different non-numeric types like `DateTime` and `DateTime64`. In previous versions bad cast occurs. In new version it will lead to exception. This closes [#26330](#). [#27682 \(alexey-milovidov\)](#).
- Fix postgresql table function resulting in non-closing connections. Closes [#26088](#). [#27662 \(Kseniia Sumarokova\)](#).
- Fixed another case of `Unexpected merged part ... intersecting drop range ...` error. [#27656 \(tavplubix\)](#).
- Fix an error with aliased column in `Distributed` table. [#27652 \(Vladimir C\)](#).
- After setting `max_memory_usage*` to non-zero value it was not possible to reset it back to 0 (unlimited). It's fixed. [#27638 \(tavplubix\)](#).
- Fixed underflow of the time value when constructing it from components. Closes [#27193](#). [#27605 \(Vasily Nemkov\)](#).
- Fix crash during projection materialization when some parts contain missing columns. This fixes [#27512](#). [#27528 \(Amos Bird\)](#).
- fix metric `BackgroundMessageBrokerSchedulePoolTask`, maybe mistyped. [#27452 \(Ben\)](#).
- Fix distributed queries with zero shards and aggregation. [#27427 \(Azat Khuzhin\)](#).
- Compatibility when `/proc/meminfo` does not contain KB suffix. [#27361 \(Mike Kot\)](#).
- Fix incorrect result for query with row-level security, `PREWHERE` and `LowCardinality` filter. Fixes [#27179](#). [#27329 \(Nikolai Kochetov\)](#).
- Fixed incorrect validation of partition id for `MergeTree` tables that created with old syntax. [#27328 \(tavplubix\)](#).
- Fix MySQL protocol when using parallel formats (CSV / TSV). [#27326 \(Raúl Marín\)](#).
- Fix `Cannot find column` error for queries with sampling. Was introduced in [#24574](#). Fixes [#26522](#). [#27301 \(Nikolai Kochetov\)](#).
- Fix errors like `Expected ColumnLowCardinality, gotUInt8` or `Bad cast from type DB::ColumnVector<char8_t> to DB::ColumnLowCardinality` for some queries with `LowCardinality` in `PREWHERE`. And more importantly, fix the lack of whitespace in the error message. Fixes [#23515](#). [#27298 \(Nikolai Kochetov\)](#).
- Fix `distributed_group_by_no_merge = 2` with `distributed_push_down_limit = 1` or `optimize_distributed_group_by_sharding_key = 1` with `LIMIT BY` and `LIMIT OFFSET`. [#27249 \(Azat Khuzhin\)](#). These are obscure combination of settings that no one is using.
- Fix mutation stuck on invalid partitions in non-replicated `MergeTree`. [#27248 \(Azat Khuzhin\)](#).
- In case of ambiguity, lambda functions prefer its arguments to other aliases or identifiers. [#27235 \(Raúl Marín\)](#).
- Fix column structure in merge join, close [#27091](#). [#27217 \(Vladimir C\)](#).

- In rare cases `system.detached_parts` table might contain incorrect information for some parts, it's fixed. Fixes [#27114](#). [#27183](#) ([tavplubix](#)).
- Fix uninitialized memory in functions `multiSearch*` with empty array, close [#27169](#). [#27181](#) ([Vladimir C.](#)).
- Fix synchronization in GRPCServer. This PR fixes [#27024](#). [#27064](#) ([Vitaly Baranov](#)).
- Fixed `cache`, `complex_key_cache`, `ssd_cache`, `complex_key_ssd_cache` configuration parsing. Options `allow_expired_keys`, `max_update_queue_size`, `update_queue_push_timeout_milliseconds`, `query_wait_timeout_milliseconds` were not parsed for dictionaries with non `cache` type. [#27032](#) ([Maksim Kita](#)).
- Fix possible mutation stack due to race with `DROP_RANGE`. [#27002](#) ([Azat Khuzhin](#)).
- Now partition ID in queries like `ALTER TABLE ... PARTITION ID xxx` validates for correctness. Fixes [#25718](#). [#26963](#) ([alesapin](#)).
- Fix "Unknown column name" error with multiple JOINS in some cases, close [#26899](#). [#26957](#) ([Vladimir C.](#)).
- Fix reading of custom TLDs (stops processing with lower buffer or bigger file). [#26948](#) ([Azat Khuzhin](#)).
- Fix error `Missing columns: 'xxx'` when `DEFAULT` column references other non materialized column without `DEFAULT` expression. Fixes [#26591](#). [#26900](#) ([alesapin](#)).
- Fix loading of dictionary keys in `library-bridge` for `library` dictionary source. [#26834](#) ([Kseniia Sumarokova](#)).
- Aggregate function parameters might be lost when applying some combinator causing exceptions like `Conversion from AggregateFunction(topKArray, Array(String)) to AggregateFunction(topKArray(10), Array(String)) is not supported`. It's fixed. Fixes [#26196](#) and [#26433](#). [#26814](#) ([tavplubix](#)).
- Add `event_time_microseconds` value for `REMOVE_PART` in `system.part_log`. In previous versions is was not set. [#26720](#) ([Azat Khuzhin](#)).
- Do not remove data on ReplicatedMergeTree table shutdown to avoid creating data to metadata inconsistency. [#26716](#) ([nvartolomei](#)).
- Sometimes `SET ROLE` could work incorrectly, this PR fixes that. [#26707](#) ([Vitaly Baranov](#)).
- Some fixes for parallel formatting (<https://github.com/ClickHouse/ClickHouse/issues/26694>). [#26703](#) ([Raúl Marín](#)).
- Fix potential `nullptr` dereference in window functions. This fixes [#25276](#). [#26668](#) ([Alexander Kuzmenkov](#)).
- Fix clickhouse-client history file conversion (when upgrading from the format of 3 years old version of clickhouse-client) if file is empty. [#26589](#) ([Azat Khuzhin](#)).
- Fix incorrect function names of `groupBitmapAnd/Or/Xor` (can be displayed in some occasions). This fixes. [#26557](#) ([Amos Bird](#)).
- Update `chown` cmd check in clickhouse-server docker entrypoint. It fixes the bug that cluster pod restart failed (or timeout) on kubernetes. [#26545](#) ([Ky Li](#)).
- Fix crash in `RabbitMQ` shutdown in case `RabbitMQ` setup was not started. Closes [#26504](#). [#26529](#) ([Kseniia Sumarokova](#)).
- Fix issues with `CREATE DICTIONARY` query if dictionary name or database name was quoted. Closes [#26491](#). [#26508](#) ([Maksim Kita](#)).

- Fix broken column name resolution after rewriting column aliases. This fixes #26432. #26475 (Amos Bird).
- Fix some fuzzed msan crash. Fixes #22517. #26428 (Nikolai Kochetov).
- Fix infinite non joined block stream in `partial_merge_join` close #26325. #26374 (Vladimir C).
- Fix possible crash when login as dropped user. This PR fixes #26073. #26363 (Vitaly Baranov).
- Fix `optimize_distributed_group_by_sharding_key` for multiple columns (leads to incorrect result w/ `optimize_skip_unused_shards=1/allow_nondeterministic_optimize_skip_unused_shards=1` and multiple columns in sharding key expression). #26353 (Azat Khuzhin).
- Fixed rare bug in lost replica recovery that may cause replicas to diverge. #26321 (tavplubix).
- Fix zstd decompression (for import/export in zstd framing format that is unrelated to tables data) in case there are escape sequences at the end of internal buffer. Closes #26013. #26314 (Kseniia Sumarokova).
- Fix logical error on join with totals, close #26017. #26250 (Vladimir C).
- Remove excessive newline in `thread_name` column in `system.stack_trace` table. This fixes #24124. #26210 (alexey-milovidov).
- Fix potential crash if more than one `untuple` expression is used. #26179 (alexey-milovidov).
- Don't throw exception in `toString` for Nullable Enum if Enum does not have a value for zero, close #25806. #26123 (Vladimir C).
- Fixed incorrect `sequence_id` in MySQL protocol packets that ClickHouse sends on exception during query execution. It might cause MySQL client to reset connection to ClickHouse server. Fixes #21184. #26051 (tavplubix).
- Fix for the case that `cutToFirstSignificantSubdomainCustom()/cutToFirstSignificantSubdomainCustomWithWWW()/firstSignificantSubdomainCustom()` returns incorrect type for consts, and hence `optimize_skip_unused_shards` does not work:. #26041 (Azat Khuzhin).
- Fix possible mismatched header when using normal projection with prewhere. This fixes #26020. #26038 (Amos Bird).
- Fix sharding\_key from column w/o function for remote() (before `select * from remote('127.1', system.one, dummy)` leads to Unknown column: `dummy`, there are only columns `.error`). #25824 (Azat Khuzhin).
- Fixed Not found column ... and Missing column ... errors when selecting from `MaterializeMySQL`. Fixes #23708, #24830, #25794. #25822 (tavplubix).
- Fix `optimize_skip_unused_shards_rewrite_in` for non-UInt64 types (may select incorrect shards eventually or throw Cannot infer type of an empty tuple or Function tuple requires at least one argument). #25798 (Azat Khuzhin).

## Build/Testing/Packaging Improvement

- Now we ran stateful and stateless tests in random timezones. Fixes #12439. Reading String as DateTime and writing DateTime as String in Protobuf format now respect timezone. Reading UInt16 as DateTime in Arrow and Parquet formats now treat it as Date and then converts to DateTime with respect to DateTime's timezone, because Date is serialized in Arrow and Parquet as UInt16. GraphiteMergeTree now respect time zone for rounding of times. Fixes #5098. Author: @alexey-milovidov. #15408 (alesapin).
- `clickhouse-test` supports SQL tests with `Jinja2` templates. #26579 (Vladimir C).

- Add support for build with clang-13. This closes #27705. #27714 (alexey-milovidov). #27777 (Sergei Semin).
- Add CMake options to build with or without specific CPU instruction set. This is for #17469 and #27509. #27508 (alexey-milovidov).
- Fix linking of auxiliar programs when using dynamic libraries. #26958 (Raúl Marín).
- Update RocksDB to 2021-07-16 master. #26411 (alexey-milovidov).

## ClickHouse release v21.8, 2021-08-12

### Upgrade Notes

- New version is using `Map` data type for system logs tables (`system.query_log`, `system.query_thread_log`, `system.processes`, `system.opentelemetry_span_log`). These tables will be auto-created with new data types. Virtual columns are created to support old queries. Closes #18698. #23934, #25773 (hexiaoting, sundy-li, Maksim Kita). If you want to *downgrade* from version 21.8 to older versions, you will need to cleanup system tables with logs manually. Look at `/var/lib/clickhouse/data/system/*_log`.

### New Features

- Add support for a part of SQL/JSON standard. #24148 (l1tsolaiki, Kseniia Sumarokova).
- Collect common system metrics (in `system.asynchronous_metrics` and `system.asynchronous_metric_log`) on CPU usage, disk usage, memory usage, IO, network, files, load average, CPU frequencies, thermal sensors, EDAC counters, system uptime; also added metrics about the scheduling jitter and the time spent collecting the metrics. It works similar to atop in ClickHouse and allows access to monitoring data even if you have no additional tools installed. Close #9430. #24416 (alexey-milovidov, Yegor Levankov).
- Add MaterializedPostgreSQL table engine and database engine. This database engine allows replicating a whole database or any subset of database tables. #20470 (Kseniia Sumarokova).
- Add new functions `leftPad()`, `rightPad()`, `leftPadUTF8()`, `rightPadUTF8()`. #26075 (Vitaly Baranov).
- Add the `FIRST` keyword to the `ADD INDEX` command to be able to add the index at the beginning of the indices list. #25904 (xjewer).
- Introduce `system.data_skipping_indices` table containing information about existing data skipping indices. Close #7659. #25693 (Dmitry Novik).
- Add `bin/unbin` functions. #25609 (zhaoyu).
- Support `Map` and `UInt128`, `Int128`, `UInt256`, `Int256` types in `mapAdd` and `mapSubtract` functions. #25596 (Ildus Kurbangaliev).
- Support `DISTINCT ON (columns)` expression, close #25404. #25589 (Zijie Lu).
- Add an ability to reset a custom setting to default and remove it from the table's metadata. It allows rolling back the change without knowing the system/config's default. Closes #14449. #17769 (xjewer).
- Render pipelines as graphs in Web UI if `EXPLAIN PIPELINE graph = 1` query is submitted. #26067 (alexey-milovidov).

### Performance Improvements

- Compile aggregate functions. Use option `compile_aggregate_expressions` to enable it. #24789 (Maksim Kita).

- Improve latency of short queries that require reading from tables with many columns. #26371 (Anton Popov).

## Improvements

- Use Map data type for system logs tables (system.query\_log, system.query\_thread\_log, system.processes, system.opentelemetry\_span\_log). These tables will be auto-created with new data types. Virtual columns are created to support old queries. Closes #18698. #23934, #25773 (hexiaoting, sundy-li, Maksim Kita).
- For a dictionary with a complex key containing only one attribute, allow not wrapping the key expression in tuple for functions dictGet, dictHas. #26130 (Maksim Kita).
- Implement function bin/hex from AggregateFunction states. #26094 (zhaoyu).
- Support arguments of UUID type for empty and notEmpty functions. UUID is empty if it is all zeros (nil UUID). Closes #3446. #25974 (zhaoyu).
- Add support for SET SQL\_SELECT\_LIMIT in MySQL protocol. Closes #17115. #25972 (Kseniia Sumarokova).
- More instrumentation for network interaction: add counters for recv/send bytes; add gauges for recvs/sends. Added missing documentation. Close #5897. #25962 (alexey-milovidov).
- Add setting optimize\_move\_to\_prewhere\_if\_final. If query has FINAL, the optimization move\_to\_prewhere will be enabled only if both optimize\_move\_to\_prewhere and optimize\_move\_to\_prewhere\_if\_final are enabled. Closes #8684. #25940 (Kseniia Sumarokova).
- Allow complex quoted identifiers of JOINed tables. Close #17861. #25924 (alexey-milovidov).
- Add support for Unicode (e.g. Chinese, Cyrillic) components in Nested data types. Close #25594. #25923 (alexey-milovidov).
- Allow quantiles\* functions to work with aggregate\_functions\_null\_for\_empty. Close #25892. #25919 (alexey-milovidov).
- Allow parameters for parametric aggregate functions to be arbitrary constant expressions (e.g., `1 + 2`), not just literals. It also allows using the query parameters (in parameterized queries like `{param:UInt8}`) inside parametric aggregate functions. Closes #11607. #25910 (alexey-milovidov).
- Correctly throw the exception on the attempt to parse an invalid Date. Closes #6481. #25909 (alexey-milovidov).
- Support for multiple includes in configuration. It is possible to include users configuration, remote server configuration from multiple sources. Simply place `<include />` element with `from_zk`, `from_env` or `incl` attribute, and it will be replaced with the substitution. #24404 (nvartolomei).
- Support for queries with a column named "null" (it must be specified in back-ticks or double quotes) and ON CLUSTER. Closes #24035. #25907 (alexey-milovidov).
- Support LowCardinality, Decimal, and UUID for JSONExtract. Closes #24606. #25900 (Kseniia Sumarokova).
- Convert history file from `readline` format to `replxx` format. #25888 (Azat Khuzhin).
- Fix an issue which can lead to intersecting parts after `DROP PART` or background deletion of an empty part. #25884 (alesapin).
- Better handling of lost parts for ReplicatedMergeTree tables. Fixes rare inconsistencies in ReplicationQueue. Fixes #10368. #25820 (alesapin).
- Allow starting clickhouse-client with unreadable working directory. #25817 (ianton-ru).
- Fix "No available columns" error for Merge storage. #25801 (Azat Khuzhin).

- MySQL Engine now supports the exchange of column comments between MySQL and ClickHouse. #25795 (Storozhuk Kostiantyn).
- Fix inconsistent behaviour of `GROUP BY` constant on empty set. Closes #6842. #25786 (Ksenia Sumarokova).
- Cancel already running merges in partition on `DROP PARTITION` and `TRUNCATE` for `ReplicatedMergeTree`. Resolves #17151. #25684 (tavplubix).
- Support `ENUM`` data type for `MaterializeMySQL`. #25676 (Storozhuk Kostiantyn).
- Support materialized and aliased columns in `JOIN`, close #13274. #25634 (Vladimir C).
- Fix possible logical race condition between `ALTER TABLE ... DETACH` and background merges. #25605 (Azat Khuzhin).
- Make `NetworkReceiveElapsedMicroseconds` metric to correctly include the time spent waiting for data from the client to `INSERT`. Close #9958. #25602 (alexey-milovidov).
- Support `TRUNCATE TABLE` for S3 and HDFS. Close #25530. #25550 (Ksenia Sumarokova).
- Support for dynamic reloading of config to change number of threads in pool for background jobs execution (merges, mutations, fetches). #25548 (Nikita Mikhaylov).
- Allow extracting of non-string element as string using `JSONExtract`. This is for #25414. #25452 (Amos Bird).
- Support regular expression in `Database` argument for `StorageMerge`. Close #776. #25064 (flynn).
- Web UI: if the value looks like a URL, automatically generate a link. #25965 (alexey-milovidov).
- Make `sudo service clickhouse-server start` to work on systems with `systemd` like Centos 8. Close #14298. Close #17799. #25921 (alexey-milovidov).

## Bug Fixes

- Fix incorrect `SET ROLE` in some cases. #26707 (Vitaly Baranov).
- Fix potential `nullptr` dereference in window functions. Fix #25276. #26668 (Alexander Kuzmenkov).
- Fix incorrect function names of `groupBitmapAnd/Or/Xor`. Fix #26557 (Amos Bird).
- Fix crash in RabbitMQ shutdown in case RabbitMQ setup was not started. Closes #26504. #26529 (Ksenia Sumarokova).
- Fix issues with `CREATE DICTIONARY` query if dictionary name or database name was quoted. Closes #26491. #26508 (Maksim Kita).
- Fix broken name resolution after rewriting column aliases. Fix #26432. #26475 (Amos Bird).
- Fix infinite non-joined block stream in `partial_merge_join` close #26325. #26374 (Vladimir C).
- Fix possible crash when login as dropped user. Fix #26073. #26363 (Vitaly Baranov).
- Fix `optimize_distributed_group_by_sharding_key` for multiple columns (leads to incorrect result w/ `optimize_skip_unused_shards=1/allow_nondeterministic_optimize_skip_unused_shards=1` and multiple columns in sharding key expression). #26353 (Azat Khuzhin).
- `CAST` from `Date` to `DateTime` (or `DateTime64`) was not using the timezone of the `DateTime` type. It can also affect the comparison between `Date` and `DateTime`. Inference of the common type for `Date` and `DateTime` also was not using the corresponding timezone. It affected the results of function `if` and array construction. Closes #24128. #24129 (Maksim Kita).

- Fixed rare bug in lost replica recovery that may cause replicas to diverge. #26321 (tavplubix).
- Fix zstd decompression in case there are escape sequences at the end of internal buffer. Closes #26013. #26314 (Kseniia Sumarokova).
- Fix logical error on join with totals, close #26017. #26250 (Vladimir C).
- Remove excessive newline in `thread_name` column in `system.stack_trace` table. Fix #24124. #26210 (alexey-milovidov).
- Fix `joinGet` with `LowCarinality` columns, close #25993. #26118 (Vladimir C).
- Fix possible crash in `pointInPolygon` if the setting `validate_polygons` is turned off. #26113 (alexey-milovidov).
- Fix throwing exception when iterate over non-existing remote directory. #26087 (ianton-ru).
- Fix rare server crash because of `abort` in ZooKeeper client. Fixes #25813. #26079 (alesapin).
- Fix wrong thread count estimation for right subquery join in some cases. Close #24075. #26052 (Vladimir C).
- Fixed incorrect `sequence_id` in MySQL protocol packets that ClickHouse sends on exception during query execution. It might cause MySQL client to reset connection to ClickHouse server. Fixes #21184. #26051 (tavplubix).
- Fix possible mismatched header when using normal projection with `PREWHERE`. Fix #26020. #26038 (Amos Bird).
- Fix formatting of type `Map` with integer keys to `JSON`. #25982 (Anton Popov).
- Fix possible deadlock during query profiler stack unwinding. Fix #25968. #25970 (Maksim Kita).
- Fix crash on call `dictGet()` with bad arguments. #25913 (Vitaly Baranov).
- Fixed `scram-sha-256` authentication for PostgreSQL engines. Closes #24516. #25906 (Kseniia Sumarokova).
- Fix extremely long backoff for background tasks when the background pool is full. Fixes #25836. #25893 (alesapin).
- Fix ARM exception handling with non default page size. Fixes #25512, #25044, #24901, #23183, #20221, #19703, #19028, #18391, #18121, #17994, #12483. #25854 (Maksim Kita).
- Fix sharding\_key from column w/o function for `remote()` (before `select * from remote('127.1', system.one, dummy)` leads to `Unknown column: dummy, there are only columns .error`). #25824 (Azat Khuzhin).
- Fixed Not found column ... and Missing column ... errors when selecting from `MaterializeMySQL`. Fixes #23708, #24830, #25794. #25822 (tavplubix).
- Fix `optimize_skip_unused_shards_rewrite_in` for non-UInt64 types (may select incorrect shards eventually or throw Cannot infer type of an empty tuple or Function tuple requires at least one argument). #25798 (Azat Khuzhin).
- Fix rare bug with `DROP PART` query for `ReplicatedMergeTree` tables which can lead to error message `Unexpected merged part intersecting drop range`. #25783 (alesapin).
- Fix bug in `TTL` with `GROUP BY` expression which refuses to execute `TTL` after first execution in part. #25743 (alesapin).
- Allow `StorageMerge` to access tables with aliases. Closes #6051. #25694 (Kseniia Sumarokova).

- Fix slow dict join in some cases, close #24209. #25618 (Vladimir C).
- Fix ALTER MODIFY COLUMN of columns, which participates in TTL expressions. #25554 (Anton Popov).
- Fix assertion in PREWHERE with non-UInt8 type, close #19589. #25484 (Vladimir C).
- Fix some fuzzed msan crash. Fixes #22517. #26428 (Nikolai Kochetov).
- Update chown cmd check in clickhouse-server docker entrypoint. It fixes error 'cluster pod restart failed (or timeout)' on kubernetes. #26545 (Ky Li).

## ClickHouse release v21.7, 2021-07-09

### Backward Incompatible Change

- Improved performance of queries with explicitly defined large sets. Added compatibility setting `legacy_column_name_of_tuple_literal`. It makes sense to set it to `true`, while doing rolling update of cluster from version lower than 21.7 to any higher version. Otherwise distributed queries with explicitly defined sets at `IN` clause may fail during update. #25371 (Anton Popov).
- Forward/backward incompatible change of maximum buffer size in clickhouse-keeper (an experimental alternative to ZooKeeper). Better to do it now (before production), than later. #25421 (alesapin).

### New Feature

- Support configuration in YAML format as alternative to XML. This closes #3607. #21858 (BoloniniD).
- Provides a way to restore replicated table when the data is (possibly) present, but the ZooKeeper metadata is lost. Resolves #13458. #13652 (Mike Kot).
- Support structs and maps in Arrow/Parquet/ORC and dictionaries in Arrow input/output formats. Present new setting `output_format_arrow_low_cardinality_as_dictionary`. #24341 (Kruglov Pavel).
- Added support for `Array` type in dictionaries. #25119 (Maksim Kita).
- Added function `bitPositionsToArray`. Closes #23792. Author [Kevin Wan] (@MaxWk). #25394 (Maksim Kita).
- Added function `dateName` to return names like 'Friday' or 'April'. Author [Daniil Kondratyev] (@dankondr). #25372 (Maksim Kita).
- Add `toJSONString` function to serialize columns to their JSON representations. #25164 (Amos Bird).
- Now `query_log` has two new columns: `initial_query_start_time`, `initial_query_start_time_microsecond` that record the starting time of a distributed query if any. #25022 (Amos Bird).
- Add aggregate function `segmentLengthSum`. #24250 (flynn).
- Add a new boolean setting `prefer_global_in_and_join` which defaults all IN/JOIN as GLOBAL IN/JOIN. #23434 (Amos Bird).
- Support ALTER DELETE queries for Join table engine. #23260 (foolchi).
- Add `quantileBFloat16` aggregate function as well as the corresponding `quantilesBFloat16` and `medianBFloat16`. It is very simple and fast quantile estimator with relative error not more than 0.390625%. This closes #16641. #23204 (Ivan Novitskiy).
- Implement `sequenceNextNode()` function useful for flow analysis. #19766 (achimbab).

### Experimental Feature

- Add support for virtual filesystem over HDFS. #11058 (overshov) (Ksenia Sumarokova).

- Now clickhouse-keeper (an experimental alternative to ZooKeeper) supports ZooKeeper-like digest ACLs. #24448 (alesapin).

## Performance Improvement

- Added optimization that transforms some functions to reading of subcolumns to reduce amount of read data. E.g., statement `col IS NULL` is transformed to reading of subcolumn `col.null`. Optimization can be enabled by setting `optimize_functions_to_subcolumns` which is currently off by default. #24406 (Anton Popov).
- Rewrite more columns to possible alias expressions. This may enable better optimization, such as projections. #24405 (Amos Bird).
- Index of type `bloom_filter` can be used for expressions with `hasAny` function with constant arrays. This closes: #24291. #24900 (Vasily Nemkov).
- Add exponential backoff to reschedule read attempt in case RabbitMQ queues are empty. (ClickHouse has support for importing data from RabbitMQ). Closes #24340. #24415 (Kseniia Sumarokova).

## Improvement

- Allow to limit bandwidth for replication. Add two Replicated\*MergeTree settings: `max_replicated_fetches_network_bandwidth` and `max_replicated_sends_network_bandwidth` which allows to limit maximum speed of replicated fetches/sends for table. Add two server-wide settings (in `default` user profile): `max_replicated_fetches_network_bandwidth_for_server` and `max_replicated_sends_network_bandwidth_for_server` which limit maximum speed of replication for all tables. The settings are not followed perfectly accurately. Turned off by default. Fixes #1821. #24573 (alesapin).
- Resource constraints and isolation for ODBC and Library bridges. Use separate `clickhouse-bridge` group and user for bridge processes. Set `oom_score_adj` so the bridges will be first subjects for OOM killer. Set set maximum RSS to 1 GiB. Closes #23861. #25280 (Kseniia Sumarokova).
- Add standalone `clickhouse-keeper` symlink to the main `clickhouse` binary. Now it's possible to run coordination without the main `clickhouse` server. #24059 (alesapin).
- Use global settings for query to VIEW. Fixed the behavior when queries to `VIEW` use local settings, that leads to errors if setting on `CREATE VIEW` and `SELECT` were different. As for now, `VIEW` won't use these modified settings, but you can still pass additional settings in `SETTINGS` section of `CREATE VIEW` query. Close #20551. #24095 (Vladimir).
- On server start, parts with incorrect partition ID would not be ever removed, but always detached. #25070. #25166 (Nikolai Kochetov).
- Increase size of background schedule pool to 128 (`background_schedule_pool_size` setting). It allows avoiding replication queue hung on slow zookeeper connection. #25072 (alesapin).
- Add merge tree setting `max_parts_to_merge_at_once` which limits the number of parts that can be merged in the background at once. Doesn't affect `OPTIMIZE FINAL` query. Fixes #1820. #24496 (alesapin).
- Allow `NOT IN` operator to be used in partition pruning. #24894 (Amos Bird).
- Recognize IPv4 addresses like `127.0.1.1` as local. This is controversial and closes #23504. Michael Filimonov will test this feature. #24316 (alexey-milovidov).
- ClickHouse database created with MaterializeMySQL (it is an experimental feature) now contains all column comments from the MySQL database that materialized. #25199 (Storozhuk Kostiantyn).

- Add settings (`connection_auto_close`/`connection_max_tries`/`connection_pool_size`) for MySQL storage engine. #24146 (Azat Khuzhin).
- Improve startup time of Distributed engine. #25663 (Azat Khuzhin).
- Improvement for Distributed tables. Drop replicas from dirname for `internal_replication=true` (allows `INSERT` into Distributed with cluster from any number of replicas, before only 15 replicas was supported, everything more will fail with `ENAMETOOLONG` while creating directory for async blocks). #25513 (Azat Khuzhin).
- Added support `Interval` type for `LowCardinality`. It is needed for intermediate values of some expressions. Closes #21730. #25410 (Vladimir).
- Add `==` operator on time conditions for `sequenceMatch` and `sequenceCount` functions. For eg: `sequenceMatch('(?1)(?t==1)(?2)')(time, data = 1, data = 2)`. #25299 (Christophe Kalenzaga).
- Add settings `http_max_fields`, `http_max_field_name_size`, `http_max_field_value_size`. #25296 (Ivan).
- Add support for function `if` with `Decimal` and `Int` types on its branches. This closes #20549. This closes #10142. #25283 (alexey-milovidov).
- Update prompt in `clickhouse-client` and display a message when reconnecting. This closes #10577. #25281 (alexey-milovidov).
- Correct memory tracking in aggregate function `topK`. This closes #25259. #25260 (alexey-milovidov).
- Fix `topLevelDomain` for IDN hosts (i.e. `example.pф`), before it returns empty string for such hosts. #25103 (Azat Khuzhin).
- Detect Linux kernel version at runtime (for worked nested epoll, that is required for `async_socket_for_remote/use_hedged_requests`, otherwise remote queries may stuck). #25067 (Azat Khuzhin).
- For distributed query, when `optimize_skip_unused_shards=1`, allow to skip shard with condition like `(sharding key) IN (one-element-tuple)`. (Tuples with many elements were supported. Tuple with single element did not work because it is parsed as literal). #24930 (Amos Bird).
- Improved log messages of S3 errors, no more double whitespaces in case of empty keys and buckets. #24897 (Vladimir Chebotarev).
- Some queries require multi-pass semantic analysis. Try reusing built sets for `IN` in this case. #24874 (Amos Bird).
- Respect `max_distributed_connections` for `insert_distributed_sync` (otherwise for huge clusters and sync insert it may run out of `max_thread_pool_size`). #24754 (Azat Khuzhin).
- Avoid hiding errors like `Limit for rows or bytes to read exceeded` for scalar subqueries. #24545 (nvartolomei).
- Make String-to-Int parser stricter so that `toInt64('+')` will throw. #24475 (Amos Bird).
- If `SSD_CACHE` is created with DDL query, it can be created only inside `user_files` directory. #24466 (Maksim Kita).
- PostgreSQL support for specifying non default schema for insert queries. Closes #24149. #24413 (Kseniia Sumarokova).
- Fix IPv6 addresses resolving (i.e. fixes `select * from remote('[:1]', system.one)`). #24319 (Azat Khuzhin).
- Fix trailing whitespaces in `FROM` clause with subqueries in multiline mode, and also changes the output of the queries slightly in a more human friendly way. #24151 (Azat Khuzhin).

- Improvement for Distributed tables. Add ability to split distributed batch on failures (i.e. due to memory limits, corruptions), under `distributed_directory_monitor_split_batch_on_failure` (OFF by default). #23864 (Azat Khuzhin).
- Handle column name clashes for Join table engine. Closes #20309. #23769 (Vladimir).
- Display progress for File table engine in `clickhouse-local` and on INSERT query in `clickhouse-client` when data is passed to stdin. Closes #18209. #23656 (Ksenia Sumarokova).
- Bugfixes and improvements of `clickhouse-copier`. Allow to copy tables with different (but compatible schemas). Closes #9159. Added test to copy ReplacingMergeTree. Closes #22711. Support TTL on columns and Data Skipping Indices. It simply removes it to create internal Distributed table (underlying table will have TTL and skipping indices). Closes #19384. Allow to copy MATERIALIZED and ALIAS columns. There are some cases in which it could be helpful (e.g. if this column is in PRIMARY KEY). Now it could be allowed by setting `allow_to_copy_alias_and_materialized_columns` property to true in task configuration. Closes #9177. Closes [#11007] (<https://github.com/ClickHouse/ClickHouse/issues/11007>). Closes #9514. Added a property `allow_to_drop_target_partitions` in task configuration to drop partition in original table before moving helping tables. Closes #20957. Get rid of `OPTIMIZE DEDUPLICATE` query. This hack was needed, because `ALTER TABLE MOVE PARTITION` was retried many times and plain MergeTree tables don't have deduplication. Closes #17966. Write progress to ZooKeeper node on path `task_path + /status` in JSON format. Closes #20955. Support for ReplicatedTables without arguments. Closes #24834. #23518 (Nikita Mikhaylov).
- Added sleep with backoff between read retries from S3. #23461 (Vladimir Chebotarev).
- Respect `insert_allow_materialized_columns` (allows materialized columns) for INSERT into Distributed table. #23349 (Azat Khuzhin).
- Add ability to push down LIMIT for distributed queries. #23027 (Azat Khuzhin).
- Fix zero-copy replication with several S3 volumes (Fixes #22679). #22864 (ianton-ru).
- Resolve the actual port number bound when a user requests any available port from the operating system to show it in the log message. #25569 (bnaecker).
- Fixed case, when sometimes conversion of postgres arrays resulted in String data type, not n-dimensional array, because `atndims` works incorrectly in some cases. Closes #24804. #25538 (Ksenia Sumarokova).
- Fix conversion of DateTime with timezone for MySQL, PostgreSQL, ODBC. Closes #5057. #25528 (Ksenia Sumarokova).
- Distinguish KILL MUTATION for different tables (fixes unexpected `Cancelled mutating parts` error). #25025 (Azat Khuzhin).
- Allow to declare S3 disk at root of bucket (S3 virtual filesystem is an experimental feature under development). #24898 (Vladimir Chebotarev).
- Enable reading of subcolumns (e.g. components of Tuples) for distributed tables. #24472 (Anton Popov).
- A feature for MySQL compatibility protocol: make `user` function to return correct output. Closes #25697. #25697 (sundyli).

## Bug Fix

- Improvement for backward compatibility. Use old modulo function version when used in partition key. Closes #23508. #24157 (Ksenia Sumarokova).
- Fix extremely rare bug on low-memory servers which can lead to the inability to perform merges without restart. Possibly fixes #24603. #24872 (alesapin).

- Fix extremely rare error Tagging already tagged part in replication queue during concurrent `alter move/replace partition`. Possibly fixes #22142. #24961 (alesapin).
- Fix potential crash when calculating aggregate function states by aggregation of aggregate function states of other aggregate functions (not a practical use case). See #24523. #25015 (alexey-milovidov).
- Fixed the behavior when query `SYSTEM RESTART REPLICA` or `SYSTEM SYNC REPLICA` does not finish. This was detected on server with extremely low amount of RAM. #24457 (Nikita Mikhaylov).
- Fix bug which can lead to ZooKeeper client hung inside clickhouse-server. #24721 (alesapin).
- If ZooKeeper connection was lost and replica was cloned after restoring the connection, its replication queue might contain outdated entries. Fixed failed assertion when replication queue contains intersecting virtual parts. It may rarely happen if some data part was lost. Print error in log instead of terminating. #24777 (tavplubix).
- Fix lost `WHERE` condition in expression-push-down optimization of query plan (setting `query_plan_filter_push_down = 1` by default). Fixes #25368. #25370 (Nikolai Kochetov).
- Fix bug which can lead to intersecting parts after merges with TTL: Part `all_40_40_0` is covered by `all_40_40_1` but should be merged into `all_40_41_1`. This shouldn't happen often.. #25549 (alesapin).
- On ZooKeeper connection loss `ReplicatedMergeTree` table might wait for background operations to complete before trying to reconnect. It's fixed, now background operations are stopped forcefully. #25306 (tavplubix).
- Fix error `Key expression contains comparison between incompatible types` for queries with `ARRAY JOIN` in case if array is used in primary key. Fixes #8247. #25546 (Anton Popov).
- Fix wrong totals for query `WITH TOTALS` and `WITH FILL`. Fixes #20872. #25539 (Anton Popov).
- Fix data race when querying `system.clusters` while reloading the cluster configuration at the same time. #25737 (Amos Bird).
- Fixed `No such file or directory` error on moving `Distributed` table between databases. Fixes #24971. #25667 (tavplubix).
- `REPLACE PARTITION` might be ignored in rare cases if the source partition was empty. It's fixed. Fixes #24869. #25665 (tavplubix).
- Fixed a bug in `Replicated` database engine that might rarely cause some replica to skip enqueued DDL query. #24805 (tavplubix).
- Fix null pointer dereference in `EXPLAIN AST` without query. #25631 (Nikolai Kochetov).
- Fix waiting of automatic dropping of empty parts. It could lead to full filling of background pool and stuck of replication. #23315 (Anton Popov).
- Fix restore of a table stored in S3 virtual filesystem (it is an experimental feature not ready for production). #25601 (ianton-ru).
- Fix nullptr dereference in Arrow format when using `Decimal256`. Add `Decimal256` support for Arrow format. #25531 (Kruglov Pavel).
- Fix excessive underscore before the names of the preprocessed configuration files. #25431 (Vitaly Baranov).
- A fix for `clickhouse-copier` tool: Fix segfault when sharding\_key is absent in task config for copier. #25419 (Nikita Mikhaylov).

- Fix `REPLACE` column transformer when used in DDL by correctly quoting the formatted query. This fixes #23925. #25391 (Amos Bird).
- Fix the possibility of non-deterministic behaviour of the `quantileDeterministic` function and similar. This closes #20480. #25313 (alexey-milovidov).
- Support `SimpleAggregateFunction(LowCardinality)` for `SummingMergeTree`. Fixes #25134. #25300 (Nikolai Kochetov).
- Fix logical error with exception message "Cannot sum Array/Tuple in min/maxMap". #25298 (Kruglov Pavel).
- Fix error `Bad cast from type DB::ColumnLowCardinality to DB::ColumnVector<char8_t>` for queries where `LowCardinality` argument was used for IN (this bug appeared in 21.6). Fixes #25187. #25290 (Nikolai Kochetov).
- Fix incorrect behaviour of `joinGetOrNull` with not-nullable columns. This fixes #24261. #25288 (Amos Bird).
- Fix incorrect behaviour and UBSan report in big integers. In previous versions `CAST(1e19 AS UInt128)` returned zero. #25279 (alexey-milovidov).
- Fixed an error which occurred while inserting a subset of columns using `CSVWithNames` format. Fixes #25129. #25169 (Nikita Mikhaylov).
- Do not use table's projection for `SELECT` with `FINAL`. It is not supported yet. #25163 (Amos Bird).
- Fix possible parts loss after updating up to 21.5 in case table used `UUID` in partition key. (It is not recommended to use `UUID` in partition key). Fixes #25070. #25127 (Nikolai Kochetov).
- Fix crash in query with cross join and `joined_subquery_requires_alias = 0`. Fixes #24011. #25082 (Nikolai Kochetov).
- Fix bug with constant maps in `mapContains` function that lead to error `empty column was returned by function mapContains`. Closes #25077. #25080 (Kruglov Pavel).
- Remove possibility to create tables with columns referencing themselves like `a UInt32 ALIAS a + 1` or `b UInt32 MATERIALIZED b`. Fixes #24910, #24292. #25059 (alesapin).
- Fix wrong result when using aggregate projection with *not empty* `GROUP BY` key to execute query with `GROUP BY` by empty key. #25055 (Amos Bird).
- Fix serialization of splitted nested messages in Protobuf format. This PR fixes #24647. #25000 (Vitaly Baranov).
- Fix limit/offset settings for distributed queries (ignore on the remote nodes). #24940 (Azat Khuzhin).
- Fix possible heap-buffer-overflow in Arrow format. #24922 (Kruglov Pavel).
- Fixed possible error 'Cannot read from istream at offset 0' when reading a file from DiskS3 (S3 virtual filesystem is an experimental feature under development that should not be used in production). #24885 (Pavel Kovalenko).
- Fix "Missing columns" exception when joining Distributed Materialized View. #24870 (Azat Khuzhin).
- Allow `NONE` values in postgresql compatibility protocol. Closes #22622. #24857 (Ksenia Sumarokova).
- Fix bug when exception `Mutation` was killed can be thrown to the client on mutation wait when mutation not loaded into memory yet. #24809 (alesapin).

- Fixed bug in deserialization of random generator state with might cause some data types such as `AggregateFunction(groupArraySample(N), T)` to behave in a non-deterministic way. #24538 ([tavplubix](#)).
- Disallow building uniqXXXXStates of other aggregation states. #24523 ([Raúl Marín](#)). Then allow it back by actually eliminating the root cause of the related issue. ([alexey-milovidov](#)).
- Fix usage of tuples in `CREATE .. AS SELECT` queries. #24464 ([Anton Popov](#)).
- Fix computation of total bytes in Buffer table. In current ClickHouse version `total_writes.bytes` counter decreases too much during the buffer flush. It leads to counter overflow and `totalBytes` return something around 17.44 EB some time after the flush. #24450 ([DimasKovas](#)).
- Fix incorrect information about the monotonicity of `toWeek` function. This fixes #24422 . This bug was introduced in <https://github.com/ClickHouse/ClickHouse/pull/5212> , and was exposed later by smarter partition pruner. #24446 ([Amos Bird](#)).
- When user authentication is managed by LDAP. Fixed potential deadlock that can happen during LDAP role (re)mapping, when LDAP group is mapped to a nonexistent local role. #24431 ([Denis Glazachev](#)).
- In "multipart/form-data" message consider the CRLF preceding a boundary as part of it. Fixes #23905. #24399 ([Ivan](#)).
- Fix drop partition with intersect fake parts. In rare cases there might be parts with mutation version greater than current block number. #24321 ([Amos Bird](#)).
- Fixed a bug in moving Materialized View from Ordinary to Atomic database (RENAME TABLE query). Now inner table is moved to new database together with Materialized View. Fixes #23926. #24309 ([tavplubix](#)).
- Allow empty HTTP headers. Fixes #23901. #24285 ([Ivan](#)).
- Correct processing of mutations (ALTER UPDATE/DELETE) in Memory tables. Closes #24274. #24275 ([flynn](#)).
- Make column LowCardinality property in JOIN output the same as in the input, close #23351, close #20315. #24061 ([Vladimir](#)).
- A fix for Kafka tables. Fix the bug in failover behavior when Engine = Kafka was not able to start consumption if the same consumer had an empty assignment previously. Closes #21118. #21267 ([filimonov](#)).

## Build/Testing/Packaging Improvement

- Add `darwin-aarch64` (Mac M1 / Apple Silicon) builds in CI #25560 ([Ivan](#)) and put the links to the docs and website ([alexey-milovidov](#)).
- Adds cross-platform embedding of binary resources into executables. It works on Illumos. #25146 ([bnaecker](#)).
- Add join related options to stress tests to improve fuzzing. #25200 ([Vladimir](#)).
- Enable build with s3 module in osx #25217. #25218 ([kevin wan](#)).
- Add integration test cases to cover JDBC bridge. #25047 ([Zhichun Wu](#)).
- Integration tests configuration has special treatment for dictionaries. Removed remaining dictionaries manual setup. #24728 ([Ilya Yatsishin](#)).
- Add libfuzzer tests for YAMLParser class. #24480 ([BoloniniD](#)).

- Ubuntu 20.04 is now used to run integration tests, docker-compose version used to run integration tests is updated to 1.28.2. Environment variables now take effect on docker-compose. Rework `test_dictionaries_all_layouts_separate_sources` to allow parallel run. #20393 ([Ilya Yatsishin](#)).
- Fix TOCTOU error in installation script. #25277 ([alexey-milovidov](#)).

## ClickHouse release 21.6, 2021-06-05

### Upgrade Notes

- `zstd` compression library is updated to v1.5.0. You may get messages about "checksum does not match" in replication. These messages are expected due to update of compression algorithm and you can ignore them. These messages are informational and do not indicate any kinds of undesired behaviour.
- The setting `compile_expressions` is enabled by default. Although it has been heavily tested on variety of scenarios, if you find some undesired behaviour on your servers, you can try turning this setting off.
- Values of `UUID` type cannot be compared with integer. For example, instead of writing `uuid != 0` type `uuid != '00000000-0000-0000-0000-000000000000'`.

### New Feature

- Add Postgres-like cast operator `(::)`. E.g.: `[1, 2]::Array(UInt8)`, `0.1::Decimal(4, 4)`, `number::UInt16`. #23871 ([Anton Popov](#)).
- Make big integers production ready. Add support for `UInt128` data type. Fix known issues with the `Decimal256` data type. Support big integers in dictionaries. Support `gcd/lcm` functions for big integers. Support big integers in array search and conditional functions. Support `LowCardinality(UUID)`. Support big integers in `generateRandom` table function and `clickhouse-obfuscator`. Fix error with returning `UUID` from scalar subqueries. This fixes #7834. This fixes #23936. This fixes #4176. This fixes #24018. Backward incompatible change: values of `UUID` type cannot be compared with integer. For example, instead of writing `uuid != 0` type `uuid != '00000000-0000-0000-0000-000000000000'`. #23631 ([alexey-milovidov](#)).
- Support `Array` data type for inserting and selecting data in `Arrow`, `Parquet` and `ORC` formats. #21770 ([taylor12805](#)).
- Implement table comments. Closes #23225. #23548 ([flynn](#)).
- Support creating dictionaries with DDL queries in `clickhouse-local`. Closes #22354. Added support for `DETACH DICTIONARY PERMANENTLY`. Added support for `EXCHANGE DICTIONARIES` for Atomic database engine. Added support for moving dictionaries between databases using `RENAME DICTIONARY`. #23436 ([Maksim Kita](#)).
- Add aggregate function `uniqTheta` to support `Theta Sketch` in ClickHouse. #23894. #22609 ([Ping Yu](#)).
- Add function `splitByRegexp`. #24077 ([abel-cheng](#)).
- Add function `arrayProduct` which accept an array as the parameter, and return the product of all the elements in array. Closes #21613. #23782 ([Maksim Kita](#)).
- Add `thread_name` column in `system.stack_trace`. This closes #23256. #24124 ([abel-cheng](#)).
- If `insert_null_as_default = 1`, insert default values instead of NULL in `INSERT ... SELECT` and `INSERT ... SELECT ... UNION ALL ...` queries. Closes #22832. #23524 ([Kseniia Sumarokova](#)).
- Add support for progress indication in `clickhouse-local` with `--progress` option. #23196 ([Egor Savin](#)).
- Add support for HTTP compression (determined by Content-Encoding HTTP header) in `http` dictionary source. This fixes #8912. #23946 ([FArthur-cmd](#)).

- Added `SYSTEM QUERY RELOAD MODEL`, `SYSTEM QUERY RELOAD MODELS`. Closes #18722. #23182 (Maksim Kita).
- Add setting `json` (boolean, 0 by default) for `EXPLAIN PLAN` query. When enabled, query output will be a single JSON row. It is recommended to use `TSVRaw` format to avoid unnecessary escaping. #23082 (Nikolai Kochetov).
- Add setting `indexes` (boolean, disabled by default) to `EXPLAIN PIPELINE` query. When enabled, shows used indexes, number of filtered parts and granules for every index applied. Supported for `MergeTree*` tables. #22352 (Nikolai Kochetov).
- LDAP: implemented user DN detection functionality to use when mapping Active Directory groups to ClickHouse roles. #22228 (Denis Glazachev).
- New aggregate function `deltaSumTimestamp` for summing the difference between consecutive rows while maintaining ordering during merge by storing timestamps. #21888 (Russ Frank).
- Added less secure IMDS credentials provider for S3 which works under docker correctly. #21852 (Vladimir Chebotarev).
- Add back `indexHint` function. This is for #21238. This reverts #9542. This fixes #9540. #21304 (Amos Bird).

## Experimental Feature

- Add `PROJECTION` support for `MergeTree*` tables. #20202 (Amos Bird).

## Performance Improvement

- Enable `compile_expressions` setting by default. When this setting enabled, compositions of simple functions and operators will be compiled to native code with LLVM at runtime. #8482 (Maksim Kita, alexey-milovidov). Note: if you feel in trouble, turn this option off.
- Update `re2` library. Performance of regular expressions matching is improved. Also this PR adds compatibility with gcc-11. #24196 (Raúl Marín).
- ORC input format reading by stripe instead of reading entire table into memory by once which is cost memory when file size is huge. #23102 (Chao Ma).
- Fusion of aggregate functions `sum`, `count` and `avg` in a query into single aggregate function. The optimization is controlled with the `optimize_fuse_sum_count_avg` setting. This is implemented with a new aggregate function `sumCount`. This function returns a tuple of two fields: `sum` and `count`. #21337 (hexiaoting).
- Update `zstd` to v1.5.0. The performance of compression is improved for single digits percentage. #24135 (Raúl Marín). Note: you may get messages about "checksum does not match" in replication. These messages are expected due to update of compression algorithm and you can ignore them.
- Improved performance of `Buffer` tables: do not acquire lock for `total_bytes/total_rows` for `Buffer` engine. #24066 (Azat Khuzhin).
- Preallocate support for hashed/sparse\_hashed dictionaries is returned. #23979 (Azat Khuzhin).
- Enable `async_socket_for_remote` by default (lower amount of threads in querying Distributed tables with large fanout). #23683 (Nikolai Kochetov).

## Improvement

- Add `_partition_value` virtual column to `MergeTree` table family. It can be used to prune partition in a deterministic way. It's needed to implement partition matcher for mutations. #23673 (Amos Bird).

- Added `region` parameter for S3 storage and disk. [#23846](#) ([Vladimir Chebotarev](#)).
- Allow configuring different log levels for different logging channels. Closes [#19569](#). [#23857](#) ([filimonov](#)).
- Keep default timezone on `DateTime` operations if it was not provided explicitly. For example, if you add one second to a value of `DateTime` type without timezone it will remain `DateTime` without timezone. In previous versions the value of default timezone was placed to the returned data type explicitly so it becomes `DateTime('something')`. This closes [#4854](#). [#23392](#) ([alexey-milovidov](#)).
- Allow user to specify empty string instead of database name for `MySQL` storage. Default database will be used for queries. In previous versions it was working for `SELECT` queries and not support for `INSERT` was also added. This closes [#19281](#). This can be useful working with `Sphinx` or other MySQL-compatible foreign databases. [#23319](#) ([alexey-milovidov](#)).
- Fixed `quantile(s)TDigest`. Added special handling of singleton centroids according to tdunning/t-digest 3.2+. Also a bug with over-compression of centroids in implementation of earlier version of the algorithm was fixed. [#23314](#) ([Vladimir Chebotarev](#)).
- Function `now64` now supports optional timezone argument. [#24091](#) ([Vasily Nemkov](#)).
- Fix the case when a progress bar in interactive mode in `clickhouse-client` that appear in the middle of the data may rewrite some parts of visible data in terminal. This closes [#19283](#). [#23050](#) ([alexey-milovidov](#)).
- Fix crash when memory allocation fails in `simdjson`. <https://github.com/simdjson/simdjson/pull/1567> . Mark as improvement because it's a very rare bug. [#24147](#) ([Amos Bird](#)).
- Preserve dictionaries until storage shutdown (this will avoid possible `external dictionary 'DICT' not found` errors at server shutdown during final flush of the `Buffer` engine). [#24068](#) ([Azat Khuzhin](#)).
- Flush `Buffer` tables before shutting down tables (within one database), to avoid discarding blocks due to underlying table had been already detached (and `Destination table default.a_data_01870 doesn't exist. Block of data is discarded` error in the log). [#24067](#) ([Azat Khuzhin](#)).
- Now `prefer_column_name_to_alias = 1` will also favor column names for `group by`, `having` and `order by`. This fixes [#23882](#). [#24022](#) ([Amos Bird](#)).
- Add support for `ORDER BY WITH FILL` with `DateTime64`. [#24016](#) ([kevin wan](#)).
- Enable `DateTime64` to be a version column in `ReplacingMergeTree`. [#23992](#) ([kevin wan](#)).
- Log information about OS name, kernel version and CPU architecture on server startup. [#23988](#) ([Azat Khuzhin](#)).
- Support specifying table schema for `postgresql` dictionary source. Closes [#23958](#). [#23980](#) ([Kseniia Sumarokova](#)).
- Add hints for names of `Enum` elements (suggest names in case of typos). Closes [#17112](#). [#23919](#) ([flynn](#)).
- Measure found rate (the percentage for which the value was found) for dictionaries (see `found_rate` in `system.dictionaries`). [#23916](#) ([Azat Khuzhin](#)).
- Allow to add specific queue settings via table setting `rabbitmq_queue_settings_list`. (Closes [#23737](#) and [#23918](#)). Allow user to control all RabbitMQ setup: if table setting `rabbitmq_queue_consume` is set to `1` - RabbitMQ table engine will only connect to specified queue and will not perform any RabbitMQ consumer-side setup like declaring exchange, queues, bindings. (Closes [#21757](#)). Add proper cleanup when RabbitMQ table is dropped - delete queues, which the table has declared and all bound exchanges - if they were created by the table. [#23887](#) ([Kseniia Sumarokova](#)).

- Add `broken_data_files/broken_data_compressed_bytes` into `system.distribution_queue`. Add metric for number of files for asynchronous insertion into Distributed tables that has been marked as broken (`BrokenDistributedFilesToInsert`). [#23885](#) ([Azat Khuzhin](#)).
- Querying `system.tables` does not go to ZooKeeper anymore. [#23793](#) ([Fuwang Hu](#)).
- Respect `lock_acquire_timeout_for_background_operations` for `OPTIMIZE` queries. [#23623](#) ([Azat Khuzhin](#)).
- Possibility to change S3 disk settings in runtime via new `SYSTEM RESTART DISK` SQL command. [#23429](#) ([Pavel Kovalenko](#)).
- If user applied a misconfiguration by mistakenly setting `max_distributed_connections` to value zero, every query to a `Distributed` table will throw exception with a message containing "logical error". But it's really an expected behaviour, not a logical error, so the exception message was slightly incorrect. It also triggered checks in our CI environment that ensures that no logical errors ever happen. Instead we will treat `max_distributed_connections` misconfigured to zero as the minimum possible value (one). [#23348](#) ([Azat Khuzhin](#)).
- Disable `min_bytes_to_use_mmap_io` by default. [#23322](#) ([Azat Khuzhin](#)).
- Support `LowCardinality` nullability with `join_use_nulls`, close [#15101](#). [#23237](#) ([vdimir](#)).
- Added possibility to restore `MergeTree` parts to `detached` directory for S3 disk. [#23112](#) ([Pavel Kovalenko](#)).
- Retries on HTTP connection drops in S3. [#22988](#) ([Vladimir Chebotarev](#)).
- Add settings `external_storage_max_read_rows` and `external_storage_max_write_rows` for MySQL table engine, dictionary source and MaterializeMySQL minor data fetches. [#22697](#) ([TCeason](#)).
- MaterializeMySQL (experimental feature): Previously, MySQL 5.7.9 was not supported due to SQL incompatibility. Now leave MySQL parameter verification to the MaterializeMySQL. [#23413](#) ([TCeason](#)).
- Enable reading of subcolumns for distributed tables. [#24472](#) ([Anton Popov](#)).
- Fix usage of tuples in `CREATE .. AS SELECT` queries. [#24464](#) ([Anton Popov](#)).
- Support for `Parquet` format in `Kafka` tables. [#23412](#) ([Chao Ma](#)).

## Bug Fix

- Use old modulo function version when used in partition key and primary key. Closes [#23508](#). [#24157](#) ([Kseniia Sumarokova](#)). It was a source of backward incompatibility in previous releases.
- Fixed the behavior when query `SYSTEM RESTART REPLICA` or `SYSTEM SYNC REPLICA` is being processed infinitely. This was detected on server with extremely little amount of RAM. [#24457](#) ([Nikita Mikhaylov](#)).
- Fix incorrect monotonicity of `toWeek` function. This fixes [#24422](#). This bug was introduced in [#5212](#), and was exposed later by smarter partition pruner. [#24446](#) ([Amos Bird](#)).
- Fix drop partition with intersect fake parts. In rare cases there might be parts with mutation version greater than current block number. [#24321](#) ([Amos Bird](#)).
- Fixed a bug in moving Materialized View from Ordinary to Atomic database (`RENAME TABLE` query). Now inner table is moved to new database together with Materialized View. Fixes [#23926](#). [#24309](#) ([tavplubix](#)).
- Allow empty HTTP headers in client requests. Fixes [#23901](#). [#24285](#) ([Ivan](#)).
- Set `max_threads = 1` to fix mutation fail of `Memory` tables. Closes [#24274](#). [#24275](#) ([flynn](#)).

- Fix typo in implementation of `Memory` tables, this bug was introduced at #15127. Closes #24192. #24193 (张中南).
- Fix abnormal server termination due to `HDFS` becoming not accessible during query execution. Closes #24117. #24191 (Ksenia Sumarokova).
- Fix crash on updating of `Nested` column with const condition. #24183 (hexiaoting).
- Fix race condition which could happen in RBAC under a heavy load. This PR fixes #24090, #24134,.. #24176 (Vitaly Baranov).
- Fix a rare bug that could lead to a partially initialized table that can serve write requests (insert/alter/so on). Now such tables will be in readonly mode. #24122 (alesapin).
- Fix an issue: `EXPLAIN PIPELINE` with `SELECT xxx FINAL` showed a wrong pipeline. (hexiaoting).
- Fixed using const `DateTime` value vs `DateTime64` column in `WHERE`. #24100 (Vasily Nemkov).
- Fix crash in merge JOIN, closes #24010. #24013 (vdimir).
- Some `ALTER PARTITION` queries might cause `Part A` intersects previous part `B` and Unexpected merged part `C` intersecting drop range `D` errors in replication queue. It's fixed. Fixes #23296. #23997 (tavplubix).
- Fix SIGSEGV for external GROUP BY and overflow row (i.e. queries like `SELECT FROM GROUP BY WITH TOTALS SETTINGS max_bytes_before_external_group_by>0, max_rows_to_group_by>0, group_by_overflow_mode='any', totals_mode='before_having'`). #23962 (Azat Khuzhin).
- Fix keys metrics accounting for `CACHE` dictionary with duplicates in the source (leads to `DictCacheKeysRequestedMiss` overflows). #23929 (Azat Khuzhin).
- Fix implementation of connection pool of PostgreSQL engine. Closes #23897. #23909 (Ksenia Sumarokova).
- Fix `distributed_group_by_no_merge = 2` with GROUP BY and aggregate function wrapped into regular function (had been broken in #23546). Throw exception in case of someone trying to use `distributed_group_by_no_merge = 2` with window functions. Disable `optimize_distributed_group_by_sharding_key` for queries with window functions. #23906 (Azat Khuzhin).
- A fix for `s3` table function: better handling of HTTP errors. Response bodies of HTTP errors were being ignored earlier. #23844 (Vladimir Chebotarev).
- A fix for `s3` table function: better handling of URI's. Fixed an incompatibility with URLs containing + symbol, data with such keys could not be read previously. #23822 (Vladimir Chebotarev).
- Fix error `Can't initialize pipeline with empty pipe` for queries with `GLOBAL IN/JOIN` and `use_hedged_requests`. Fixes #23431. #23805 (Nikolai Kochetov).
- Fix `CLEAR COLUMN` does not work when it is referenced by materialized view. Close #23764. #23781 (flynn).
- Fix heap use after free when reading from HDFS if `Values` format is used. #23761 (Ksenia Sumarokova).
- Avoid possible "Cannot schedule a task" error (in case some exception had been occurred) on INSERT into Distributed. #23744 (Azat Khuzhin).
- Fixed a bug in recovery of staled ReplicatedMergeTree replica. Some metadata updates could be ignored by staled replica if `ALTER` query was executed during downtime of the replica. #23742 (tavplubix).
- Fix a bug with `Join` and `WITH TOTALS`, close #17718. #23549 (vdimir).

- Fix possible `Block` structure mismatch error for queries with `UNION` which could possibly happen after filter-pushdown optimization. Fixes #23029. #23359 (Nikolai Kochetov).
- Add type conversion when the setting `optimize_skip_unused_shards_rewrite_in` is enabled. This fixes MSan report. #23219 (Azat Khuzhin).
- Add a missing check when updating nested subcolumns, close issue: #22353. #22503 (hexiaoting).

## Build/Testing/Packaging Improvement

- Support building on Illumos. #24144. Adds support for building on Solaris-derived operating systems. #23746 (bnaecker).
- Add more benchmarks for hash tables, including the Swiss Table from Google (that appeared to be slower than ClickHouse hash map in our specific usage scenario). #24111 (Maksim Kita).
- Update librdkafka 1.6.0-RC3 to 1.6.1. #23874 (filimonov).
- Always enable `asynchronous-unwind-tables` explicitly. It may fix query profiler on AArch64. #23602 (alexey-milovidov).
- Avoid possible build dependency on locale and filesystem order. This allows reproducible builds. #23600 (alexey-milovidov).
- Remove a source of nondeterminism from build. Now builds at different point of time will produce byte-identical binaries. Partially addressed #22113. #23559 (alexey-milovidov).
- Add simple tool for benchmarking (Zoo)Keeper. #23038 (alesapin).

## ClickHouse release 21.5, 2021-05-20

### Backward Incompatible Change

- Change comparison of integers and floating point numbers when integer is not exactly representable in the floating point data type. In new version comparison will return false as the rounding error will occur. Example: `9223372036854775808.0 != 9223372036854775808`, because the number `9223372036854775808` is not representable as floating point number exactly (and `9223372036854775808.0` is rounded to `9223372036854776000.0`). But in previous version the comparison will return as the numbers are equal, because if the floating point number `9223372036854776000.0` get converted back to `UInt64`, it will yield `9223372036854775808`. For the reference, the Python programming language also treats these numbers as equal. But this behaviour was dependend on CPU model (different results on AMD64 and AArch64 for some out-of-range numbers), so we make the comparison more precise. It will treat int and float numbers equal only if int is represented in floating point type exactly. #22595 (alexey-milovidov).
- Remove support for `argMin` and `argMax` for single `Tuple` argument. The code was not memory-safe. The feature was added by mistake and it is confusing for people. These functions can be reintroduced under different names later. This fixes #22384 and reverts #17359. #23393 (alexey-milovidov).

### New Feature

- Added functions `dictGetChildren(dictionary, key)`, `dictGetDescendants(dictionary, key, level)`. Function `dictGetChildren` return all children as an array of indexes. It is a inverse transformation for `dictGetHierarchy`. Function `dictGetDescendants` return all descendants as if `dictGetChildren` was applied `level` times recursively. Zero `level` value is equivalent to infinity. Improved performance of `dictGetHierarchy`, `dictIsIn` functions. Closes #14656. #22096 (Maksim Kita).
- Added function `dictGetOrNull`. It works like `dictGet`, but return `Null` in case key was not found in dictionary. Closes #22375. #22413 (Maksim Kita).

- Added a table function `s3Cluster`, which allows to process files from `s3` in parallel on every node of a specified cluster. [#22012 \(Nikita Mikhaylov\)](#).
- Added support for replicas and shards in MySQL/PostgreSQL table engine / table function. You can write `SELECT * FROM mysql('host{1,2}-{1|2}', ...)`. Closes [#20969](#). [#22217 \(Ksenia Sumarokova\)](#).
- Added `ALTER TABLE ... FETCH PART ...query`. It's similar to `FETCH PARTITION`, but fetches only one part. [#22706 \(turbo jason\)](#).
- Added a setting `max_distributed_depth` that limits the depth of recursive queries to `Distributed` tables. Closes [#20229](#). [#21942 \(flynn\)](#).

## Performance Improvement

- Improved performance of `intDiv` by dynamic dispatch for AVX2. This closes [#22314](#). [#23000 \(alexey-milovidov\)](#).
- Improved performance of reading from `ArrowStream` input format for sources other than local file (e.g. URL). [#22673 \(nvartolomei\)](#).
- Disabled compression by default when interacting with localhost (with `clickhouse-client` or server to server with distributed queries) via native protocol. It may improve performance of some import/export operations. This closes [#22234](#). [#22237 \(alexey-milovidov\)](#).
- Exclude values that does not belong to the shard from right part of IN section for distributed queries (under `optimize_skip_unused_shards_rewrite_in`, enabled by default, since it still requires `optimize_skip_unused_shards`). [#21511 \(Azat Khuzhin\)](#).
- Improved performance of reading a subset of columns with File-like table engine and column-oriented format like Parquet, Arrow or ORC. This closes [#issue:20129](#). [#21302 \(keenwolf\)](#).
- Allow to move more conditions to `PREWHERE` as it was before version 21.1 (adjustment of internal heuristics). Insufficient number of moved conditions could lead to worse performance. [#23397 \(Anton Popov\)](#).
- Improved performance of ODBC connections and fixed all the outstanding issues from the backlog. Using `nanodbc` library instead of `Poco::ODBC`. Closes [#9678](#). Add support for `DateTime64` and `Decimal*` for ODBC table engine. Closes [#21961](#). Fixed issue with cyrillic text being truncated. Closes [#16246](#). Added connection pools for odbc bridge. [#21972 \(Ksenia Sumarokova\)](#).

## Improvement

- Increase `max_uri_size` (the maximum size of URL in HTTP interface) to 1 MiB by default. This closes [#21197](#). [#22997 \(alexey-milovidov\)](#).
- Set `background_fetches_pool_size` to 8 that is better for production usage with frequent small insertions or slow ZooKeeper cluster. [#22945 \(alexey-milovidov\)](#).
- FlatDictionary added `initial_array_size`, `max_array_size` options. [#22521 \(Maksim Kita\)](#).
- Add new setting `non_replicated_deduplication_window` for non-replicated MergeTree inserts deduplication. [#22514 \(alesapin\)](#).
- Update paths to the `CatBoost` model configs in config reloading. [#22434 \(Kruglov Pavel\)](#).
- Added `Decimal256` type support in dictionaries. `Decimal256` is experimental feature. Closes [#20979](#). [#22960 \(Maksim Kita\)](#).
- Enabled `async_socket_for_remote` by default (using less amount of OS threads for distributed queries). [#23683 \(Nikolai Kochetov\)](#).

- Fixed `quantile(s)TDigest`. Added special handling of singleton centroids according to `tdunning/t-digest` 3.2+. Also a bug with over-compression of centroids in implementation of earlier version of the algorithm was fixed. [#23314](#) ([Vladimir Chebotarev](#)).
- Make function name `unhex` case insensitive for compatibility with MySQL. [#23229](#) ([alexey-milovidov](#)).
- Implement functions `arrayHasAny`, `arrayHasAll`, `has`, `indexOf`, `countEqual` for generic case when types of array elements are different. In previous versions the functions `arrayHasAny`, `arrayHasAll` returned false and `has`, `indexOf`, `countEqual` thrown exception. Also add support for `Decimal` and big integer types in functions `has` and similar. This closes [#20272](#). [#23044](#) ([alexey-milovidov](#)).
- Raised the threshold on max number of matches in result of the function `extractAllGroupsHorizontal`. [#23036](#) ([Vasily Nemkov](#)).
- Do not perform `optimize_skip_unused_shards` for cluster with one node. [#22999](#) ([Azat Khuzhin](#)).
- Added ability to run `clickhouse-keeper` (experimental drop-in replacement to ZooKeeper) with SSL. Config settings `keeper_server.tcp_port_secure` can be used for secure interaction between client and keeper-server. `keeper_server.raft_configuration.secure` can be used to enable internal secure communication between nodes. [#22992](#) ([alesapin](#)).
- Added ability to flush buffer only in background for `Buffer` tables. [#22986](#) ([Azat Khuzhin](#)).
- When selecting from `MergeTree` table with `NULL` in `WHERE` condition, in rare cases, exception was thrown. This closes [#20019](#). [#22978](#) ([alexey-milovidov](#)).
- Fix error handling in Poco HTTP Client for AWS. [#22973](#) ([kreuzerkrieg](#)).
- Respect `max_part_removal_threads` for `ReplicatedMergeTree`. [#22971](#) ([Azat Khuzhin](#)).
- Fix obscure corner case of `MergeTree` settings `inactive_parts_to_throw_insert = 0` with `inactive_parts_to_delay_insert > 0`. [#22947](#) ([Azat Khuzhin](#)).
- `dateDiff` now works with `DateTime64` arguments (even for values outside of `DateTime` range) [#22931](#) ([Vasily Nemkov](#)).
- MaterializeMySQL (experimental feature): added an ability to replicate MySQL databases containing views without failing. This is accomplished by ignoring the views. [#22760](#) ([Christian](#)).
- Allow RBAC row policy via postgresql protocol. Closes [#22658](#). PostgreSQL protocol is enabled in configuration by default. [#22755](#) ([Kseniia Sumarokova](#)).
- Add metric to track how much time is spend during waiting for Buffer layer lock. [#22725](#) ([Azat Khuzhin](#)).
- Allow to use CTE in VIEW definition. This closes [#22491](#). [#22657](#) ([Amos Bird](#)).
- Clear the rest of the screen and show cursor in `clickhouse-client` if previous program has left garbage in terminal. This closes [#16518](#). [#22634](#) ([alexey-milovidov](#)).
- Make `round` function to behave consistently on non-x86\_64 platforms. Rounding half to nearest even (Banker's rounding) is used. [#22582](#) ([alexey-milovidov](#)).
- Correctly check structure of blocks of data that are sending by Distributed tables. [#22325](#) ([Azat Khuzhin](#)).
- Allow publishing Kafka errors to a virtual column of Kafka engine, controlled by the `kafka_handle_error_mode` setting. [#21850](#) ([fastio](#)).
- Add aliases `simpleJSONExtract/simpleJSONHas` to `visitParam/visitParamExtract{UInt, Int, Bool, Float, Raw, String}`. Fixes [#21383](#). [#21519](#) ([fastio](#)).

- Add `clickhouse-library-bridge` for library dictionary source. Closes #9502. #21509 (Ksenia Sumarokova).
- Forbid to drop a column if it's referenced by materialized view. Closes #21164. #21303 (flynn).
- Support dynamic interserver credentials (rotating credentials without downtime). #14113 (johnskopis).
- Add support for Kafka storage with `Arrow` and `ArrowStream` format messages. #23415 (Chao Ma).
- Fixed missing semicolon in exception message. The user may find this exception message unpleasant to read. #23208 (alexey-milovidov).
- Fixed missing whitespace in some exception messages about `LowCardinality` type. #23207 (alexey-milovidov).
- Some values were formatted with alignment in center in table cells in `Markdown` format. Not anymore. #23096 (alexey-milovidov).
- Remove non-essential details from suggestions in `clickhouse-client`. This closes #22158. #23040 (alexey-milovidov).
- Correct calculation of `bytes_allocated` field in `system.dictionaries` for `sparse_hashed` dictionaries. #22867 (Azat Khuzhin).
- Fixed approximate total rows accounting for reverse reading from `MergeTree`. #22726 (Azat Khuzhin).
- Fix the case when it was possible to configure dictionary with `clickhouse` source that was looking to itself that leads to infinite loop. Closes #14314. #22479 (Maksim Kita).

## Bug Fix

- Multiple fixes for hedged requests. Fixed an error `Can't initialize pipeline with empty pipe` for queries with `GLOBAL IN/JOIN` when the setting `use_hedged_requests` is enabled. Fixes #23431. #23805 (Nikolai Kochetov). Fixed a race condition in hedged connections which leads to crash. This fixes #22161. #22443 (Kruglov Pavel). Fix possible crash in case if `unknown` packet was received from remote query (with `async_socket_for_remote` enabled). Fixes #21167. #23309 (Nikolai Kochetov).
- Fixed the behavior when disabling `input_format_with_names_use_header` setting discards all the input with `CSVWithNames` format. This fixes #22406. #23202 (Nikita Mikhaylov).
- Fixed remote JDBC bridge timeout connection issue. Closes #9609. #23771 (Maksim Kita, alexey-milovidov).
- Fix the logic of initial load of `complex_key_hashed` if `update_field` is specified. Closes #23800. #23824 (Maksim Kita).
- Fixed crash when `PREWHERE` and row policy filter are both in effect with empty result. #23763 (Amos Bird).
- Avoid possible "Cannot schedule a task" error (in case some exception had been occurred) on `INSERT` into `Distributed`. #23744 (Azat Khuzhin).
- Added an exception in case of completely the same values in both samples in aggregate function `mannWhitneyUTest`. This fixes #23646. #23654 (Nikita Mikhaylov).
- Fixed server fault when inserting data through HTTP caused an exception. This fixes #23512. #23643 (Nikita Mikhaylov).
- Fixed misinterpretation of some `LIKE` expressions with escape sequences. #23610 (alexey-milovidov).
- Fixed restart / stop command hanging. Closes #20214. #23552 (filimonov).

- Fixed `COLUMNS` matcher in case of multiple JOINS in select query. Closes #22736. #23501 (Maksim Kita).
- Fixed a crash when modifying column's default value when a column itself is used as `ReplacingMergeTree`'s parameter. #23483 (hexiaoting).
- Fixed corner cases in vertical merges with `ReplacingMergeTree`. In rare cases they could lead to fails of merges with exceptions like `Incomplete granules are not allowed while blocks are granules size` #23459 (Anton Popov).
- Fixed bug that does not allow cast from empty array literal, to array with dimensions greater than 1, e.g. `CAST([] AS Array(Array(String)))`. Closes #14476. #23456 (Maksim Kita).
- Fixed a bug when `deltaSum` aggregate function produced incorrect result after resetting the counter. #23437 (Russ Frank).
- Fixed `Cannot unlink file` error on unsuccessful creation of `ReplicatedMergeTree` table with multidisk configuration. This closes #21755. #23433 (tavplubix).
- Fixed incompatible constant expression generation during partition pruning based on virtual columns. This fixes [https://github.com/ClickHouse/ClickHouse/pull/21401#discussion\\_r611888913](https://github.com/ClickHouse/ClickHouse/pull/21401#discussion_r611888913). #23366 (Amos Bird).
- Fixed a crash when setting `join_algorithm` is set to 'auto' and Join is performed with a Dictionary. Close #23002. #23312 (Vladimir).
- Don't relax NOT conditions during partition pruning. This fixes #23305 and #21539. #23310 (Amos Bird).
- Fixed very rare race condition on background cleanup of old blocks. It might cause a block not to be deduplicated if it's too close to the end of deduplication window. #23301 (tavplubix).
- Fixed very rare (distributed) race condition between creation and removal of `ReplicatedMergeTree` tables. It might cause exceptions like `node doesn't exist` on attempt to create replicated table. Fixes #21419. #23294 (tavplubix).
- Fixed simple key dictionary from DDL creation if primary key is not first attribute. Fixes #23236. #23262 (Maksim Kita).
- Fixed reading from ODBC when there are many long column names in a table. Closes #8853. #23215 (Kseniia Sumarokova).
- MaterializeMySQL (experimental feature): fixed `Not found column` error when selecting from `MaterializeMySQL` with condition on key column. Fixes #22432. #23200 (tavplubix).
- Correct aliases handling if subquery was optimized to constant. Fixes #22924. Fixes #10401. #23191 (Maksim Kita).
- Server might fail to start if `data_type_default_nullable` setting is enabled in default profile, it's fixed. Fixes #22573. #23185 (tavplubix).
- Fixed a crash on shutdown which happened because of wrong accounting of current connections. #23154 (Vitaly Baranov).
- Fixed `Table .inner_id... doesn't exist` error when selecting from Materialized View after detaching it from Atomic database and attaching back. #23047 (tavplubix).
- Fix error `Cannot find column` in ActionsDAG result which may happen if subquery uses `untuple`. Fixes #22290. #22991 (Nikolai Kochetov).
- Fix usage of constant columns of type `Map` with nullable values. #22939 (Anton Popov).

- fixed `formatDateTime()` on `DateTime64` and "%C" format specifier fixed `toDateTime64()` for large values and non-zero scale. #22937 ([Vasily Nemkov](#)).
- Fixed a crash when using `mannWhitneyUTest` and `rankCorr` with window functions. This fixes #22728. #22876 ([Nikita Mikhaylov](#)).
- LIVE VIEW (experimental feature): fixed possible hanging in concurrent DROP/CREATE of TEMPORARY LIVE VIEW in `TemporaryLiveViewCleaner`, see. #22858 ([Vitaly Baranov](#)).
- Fixed pushdown of `HAVING` in case, when filter column is used in aggregation. #22763 ([Anton Popov](#)).
- Fixed possible hangs in Zookeeper requests in case of OOM exception. Fixes #22438. #22684 ([Nikolai Kochetov](#)).
- Fixed wait for mutations on several replicas for ReplicatedMergeTree table engines. Previously, mutation/alter query may finish before mutation actually executed on other replicas. #22669 ([alesapin](#)).
- Fixed exception for Log with nested types without columns in the SELECT clause. #22654 ([Azat Khuzhin](#)).
- Fix unlimited wait for auxiliary AWS requests. #22594 ([Vladimir Chebotarev](#)).
- Fixed a crash when client closes connection very early #22579. #22591 ([nvartolomei](#)).
- Map data type (experimental feature): fixed an incorrect formatting of function `map` in distributed queries. #22588 ([foolchi](#)).
- Fixed deserialization of empty string without newline at end of TSV format. This closes #20244. Possible workaround without version update: set `input_format_null_as_default` to zero. It was zero in old versions. #22527 ([alexey-milovidov](#)).
- Fixed wrong cast of a column of `LowCardinality` type in Merge Join algorithm. Close #22386, close #22388. #22510 ([Vladimir](#)).
- Buffer overflow (on read) was possible in `tokenbf_v1` full text index. The excessive bytes are not used but the read operation may lead to crash in rare cases. This closes #19233. #22421 ([alexey-milovidov](#)).
- Do not limit HTTP chunk size. Fixes #21907. #22322 ([Ivan](#)).
- Fixed a bug, which leads to underaggregation of data in case of enabled `optimize_aggregation_in_order` and many parts in table. Slightly improve performance of aggregation with enabled `optimize_aggregation_in_order`. #21889 ([Anton Popov](#)).
- Check if table function view is used as a column. This complements #20350. #21465 ([Amos Bird](#)).
- Fix "unknown column" error for tables with `Merge` engine in queris with `JOIN` and aggregation. Closes #18368, close #22226. #21370 ([Vladimir](#)).
- Fixed name clashes in pushdown optimization. It caused incorrect `WHERE` filtration after FULL JOIN. Close #20497. #20622 ([Vladimir](#)).
- Fixed very rare bug when quorum insert with `quorum_parallel=1` is not really "quorum" because of deduplication. #18215 ([filimonov](#) - reported, [alesapin](#) - fixed).

## Build/Testing/Packaging Improvement

- Run stateless tests in parallel in CI. #22300 ([alesapin](#)).
- Simplify debian packages. This fixes #21698. #22976 ([alexey-milovidov](#)).
- Added support for ClickHouse build on Apple M1. #21639 ([changvzb](#)).

- Fixed ClickHouse Keeper build for MacOS. #22860 ([alesapin](#)).
- Fixed some tests on AArch64 platform. #22596 ([alexey-milovidov](#)).
- Added function alignment for possibly better performance. #21431 ([Danila Kutenin](#)).
- Adjust some tests to output identical results on amd64 and aarch64 (qemu). The result was depending on implementation specific CPU behaviour. #22590 ([alexey-milovidov](#)).
- Allow query profiling only on x86\_64. See #15174 and #15638. This closes #15638. #22580 ([alexey-milovidov](#)).
- Allow building with unbundled xz (lzma) using `USE_INTERNAL_XZ_LIBRARY=OFF` CMake option. #22571 ([Kfir Itzhak](#)).
- Enable bundled `openldap` on `ppc64le` #22487 ([Kfir Itzhak](#)).
- Disable incompatible libraries (platform specific typically) on `ppc64le` #22475 ([Kfir Itzhak](#)).
- Add Jepsen test in CI for clickhouse Keeper. #22373 ([alesapin](#)).
- Build `jemalloc` with support for `heap profiling`. #22834 ([nvartolomei](#)).
- Avoid UB in `*Log` engines for rwlock unlock due to unlock from another thread. #22583 ([Azat Khuzhin](#)).
- Fixed UB by unlocking the rwlock of the TinyLog from the same thread. #22560 ([Azat Khuzhin](#)).

## ClickHouse release 21.4

### ClickHouse release 21.4.1 2021-04-12

#### Backward Incompatible Change

- The `toStartOfIntervalFunction` will align hour intervals to the midnight (in previous versions they were aligned to the start of unix epoch). For example, `toStartOfInterval(x, INTERVAL 11 HOUR)` will split every day into three intervals: `00:00:00..10:59:59`, `11:00:00..21:59:59` and `22:00:00..23:59:59`. This behaviour is more suited for practical needs. This closes #9510. #22060 ([alexey-milovidov](#)).
- `Age` and `Precision` in graphite rollup configs should increase from retention to retention. Now it's checked and the wrong config raises an exception. #21496 ([Mikhail f. Shiryaev](#)).
- Fix `cutToFirstSignificantSubdomainCustom()/firstSignificantSubdomainCustom()` returning wrong result for 3+ level domains present in custom top-level domain list. For input domains matching these custom top-level domains, the third-level domain was considered to be the first significant one. This is now fixed. This change may introduce incompatibility if the function is used in e.g. the sharding key. #21946 ([Azat Khuzhin](#)).
- Column `keys` in table `system.dictionaries` was replaced to columns `key.names` and `key.types`. Columns `key.names`, `key.types`, `attribute.names`, `attribute.types` from `system.dictionaries` table does not require dictionary to be loaded. #21884 ([Maksim Kita](#)).
- Now replicas that are processing the `ALTER TABLE ATTACH PART[ITION]` command search in their `detached` folders before fetching the data from other replicas. As an implementation detail, a new command `ATTACH_PART` is introduced in the replicated log. Parts are searched and compared by their checksums. #18978 ([Mike Kot](#)). **Note:**
- `ATTACH PART[ITION]` queries may not work during cluster upgrade.
- It's not possible to rollback to older ClickHouse version after executing `ALTER ... ATTACH` query in new version as the old servers would fail to pass the `ATTACH_PART` entry in the replicated log.

- In this version, empty `<remote_url_allow_hosts></remote_url_allow_hosts>` will block all access to remote hosts while in previous versions it did nothing. If you want to keep old behaviour and you have empty `remote_url_allow_hosts` element in configuration file, remove it. #20058 (Vladimir Chebotarev).

## New Feature

- Extended range of `DateTime64` to support dates from year 1925 to 2283. Improved support of `DateTime` around zero date (1970-01-01). #9404 (alexey-milovidov, Vasily Nemkov). Not every time and date functions are working for extended range of dates.
- Added support of Kerberos authentication for preconfigured users and HTTP requests (GSS-SPNEGO). #14995 (Denis Glazachev).
- Add `prefer_column_name_to_alias` setting to use original column names instead of aliases. It is needed to be more compatible with common databases' aliasing rules. This is for #9715 and #9887. #22044 (Amos Bird).
- Added functions `dictGetChildren(dictionary, key)`, `dictGetDescendants(dictionary, key, level)`. Function `dictGetChildren` return all children as an array of indexes. It is a inverse transformation for `dictGetHierarchy`. Function `dictGetDescendants` return all descendants as if `dictGetChildren` was applied `level` times recursively. Zero `level` value is equivalent to infinity. Closes #14656. #22096 (Maksim Kita).
- Added `executable_pool` dictionary source. Close #14528. #21321 (Maksim Kita).
- Added table function `dictionary`. It works the same way as `Dictionary` engine. Closes #21560. #21910 (Maksim Kita).
- Support `Nullable` type for `PolygonDictionary` attribute. #21890 (Maksim Kita).
- Functions `dictGet`, `dictHas` use current database name if it is not specified for dictionaries created with DDL. Closes #21632. #21859 (Maksim Kita).
- Added function `dictGetOrNull`. It works like `dictGet`, but return `Null` in case key was not found in dictionary. Closes #22375. #22413 (Maksim Kita).
- Added async update in `ComplexKeyCache`, `SSDCache`, `SSDComplexKeyCache` dictionaries. Added support for `Nullable` type in `Cache`, `ComplexKeyCache`, `SSDCache`, `SSDComplexKeyCache` dictionaries. Added support for multiple attributes fetch with `dictGet`, `dictGetOrDefault` functions. Fixes #21517. #20595 (Maksim Kita).
- Support `dictHas` function for `RangeHashedDictionary`. Fixes #6680. #19816 (Maksim Kita).
- Add function `timezoneOf` that returns the timezone name of `DateTime` or `DateTime64` data types. This does not close #9959. Fix inconsistencies in function names: add aliases `timezone` and `timeZone` as well as `toTimezone` and `toTimeZone` and `timezoneOf` and `timeZoneOf`. #22001 (alexey-milovidov).
- Add new optional clause `GRANTEES` for `CREATE/ALTER USER` commands. It specifies users or roles which are allowed to receive grants from this user on condition this user has also all required access granted with grant option. By default `GRANTEES ANY` is used which means a user with grant option can grant to anyone. Syntax: `CREATE USER ... GRANTEES {user | role | ANY | NONE} [,...] [EXCEPT {user | role} [,...]]` #21641 (Vitaly Baranov).
- Add new column `slowdowns_count` to `system.clusters`. When using hedged requests, it shows how many times we switched to another replica because this replica was responding slowly. Also show actual value of `errors_count` in `system.clusters`. #21480 (Kruglov Pavel).
- Add `_partition_id` virtual column for `MergeTree*` engines. Allow to prune partitions by `_partition_id`. Add `partitionID()` function to calculate partition id string. #21401 (Amos Bird).
- Add function `isIPAddressInRange` to test if an IPv4 or IPv6 address is contained in a given CIDR network prefix. #21329 (PHO).

- Added new SQL command `ALTER TABLE 'table_name' UNFREEZE [PARTITION 'part_expr'] WITH NAME 'backup_name'`. This command is needed to properly remove 'freezed' partitions from all disks. [#21142](#) ([Pavel Kovalenko](#)).
- Supports implicit key type conversion for JOIN. [#19885](#) ([Vladimir](#)).

## Experimental Feature

- Support `RANGE OFFSET` frame (for window functions) for floating point types. Implement `lagInFrame/leadInFrame` window functions, which are analogous to `lag/lead`, but respect the window frame. They are identical when the frame is `between unbounded preceding and unbounded following`. This closes [#5485](#). [#21895](#) ([Alexander Kuzmenkov](#)).
- Zero-copy replication for `ReplicatedMergeTree` over S3 storage. [#16240](#) ([ianton-ru](#)).
- Added possibility to migrate existing S3 disk to the schema with backup-restore capabilities. [#22070](#) ([Pavel Kovalenko](#)).

## Performance Improvement

- Supported parallel formatting in `clickhouse-local` and everywhere else. [#21630](#) ([Nikita Mikhaylov](#)).
- Support parallel parsing for `CSVWithNames` and `TSVWithNames` formats. This closes [#21085](#). [#21149](#) ([Nikita Mikhaylov](#)).
- Enable read with mmap IO for file ranges from 64 MiB (the settings `min_bytes_to_use_mmap_io`). It may lead to moderate performance improvement. [#22326](#) ([alexey-milovidov](#)).
- Add cache for files read with `min_bytes_to_use_mmap_io` setting. It makes significant (2x and more) performance improvement when the value of the setting is small by avoiding frequent mmap/munmap calls and the consequent page faults. Note that mmap IO has major drawbacks that makes it less reliable in production (e.g. hung or SIGBUS on faulty disks; less controllable memory usage). Nevertheless it is good in benchmarks. [#22206](#) ([alexey-milovidov](#)).
- Avoid unnecessary data copy when using codec `NONE`. Please note that codec `NONE` is mostly useless - it's recommended to always use compression (`LZ4` is by default). Despite the common belief, disabling compression may not improve performance (the opposite effect is possible). The `NONE` codec is useful in some cases: - when data is uncompressable; - for synthetic benchmarks. [#22145](#) ([alexey-milovidov](#)).
- Faster `GROUP BY` with small `max_rows_to_group_by` and `group_by_overflow_mode='any'`. [#21856](#) ([Nikolai Kochetov](#)).
- Optimize performance of queries like `SELECT ... FINAL ... WHERE`. Now in queries with `FINAL` it's allowed to move to `PREDWHERE` columns, which are in sorting key. [#21830](#) ([foolchi](#)).
- Improved performance by replacing `memcpy` to another implementation. This closes [#18583](#). [#21520](#) ([alexey-milovidov](#)).
- Improve performance of aggregation in order of sorting key (with enabled setting `optimize_aggregation_in_order`). [#19401](#) ([Anton Popov](#)).

## Improvement

- Add connection pool for PostgreSQL table/database engine and dictionary source. Should fix [#21444](#). [#21839](#) ([Kseniia Sumarokova](#)).
- Support non-default table schema for postgres storage/table-function. Closes [#21701](#). [#21711](#) ([Kseniia Sumarokova](#)).
- Support replicas priority for postgres dictionary source. [#21710](#) ([Kseniia Sumarokova](#)).

- Introduce a new merge tree setting `min_bytes_to_rebalance_partition_over_jbod` which allows assigning new parts to different disks of a JBOD volume in a balanced way. [#16481 \(Amos Bird\)](#).
- Added `Grant`, `Revoke` and `System` values of `query_kind` column for corresponding queries in `system.query_log`. [#21102 \(Vasily Nemkov\)](#).
- Allow customizing timeouts for HTTP connections used for replication independently from other HTTP timeouts. [#20088 \(nvartolomei\)](#).
- Better exception message in client in case of exception while server is writing blocks. In previous versions client may get misleading message like `Data compressed with different methods`. [#22427 \(alexey-milovidov\)](#).
- Fix error `Directory tmp_fetch_XXX already exists` which could happen after failed fetch part. Delete temporary fetch directory if it already exists. Fixes [#14197](#). [#22411 \(nvartolomei\)](#).
- Fix MSan report for function `range` with `UInt256` argument (support for large integers is experimental). This closes [#22157](#). [#22387 \(alexey-milovidov\)](#).
- Add `current_database` column to `system.processes` table. It contains the current database of the query. [#22365 \(Alexander Kuzmenkov\)](#).
- Add case-insensitive history search/navigation and subword movement features to `clickhouse-client`. [#22105 \(Amos Bird\)](#).
- If tuple of NULLs, e.g. `(NULL, NULL)` is on the left hand side of `IN` operator with tuples of non-NULLs on the right hand side, e.g. `SELECT (NULL, NULL) IN ((0, 0), (3, 1))` return 0 instead of throwing an exception about incompatible types. The expression may also appear due to optimization of something like `SELECT (NULL, NULL) = (8, 0) OR (NULL, NULL) = (3, 2) OR (NULL, NULL) = (0, 0) OR (NULL, NULL) = (3, 1)`. This closes [#22017](#). [#22063 \(alexey-milovidov\)](#).
- Update used version of `simdjson` to 0.9.1. This fixes [#21984](#). [#22057 \(Vitaly Baranov\)](#).
- Added case insensitive aliases for `CONNECTION_ID()` and `VERSION()` functions. This fixes [#22028](#). [#22042 \(Eugene Klimov\)](#).
- Add option `strict_increase` to `windowFunnel` function to calculate each event once (resolve [#21835](#)). [#22025 \(Vladimir\)](#).
- If partition key of a `MergeTree` table does not include `Date` or `DateTime` columns but includes exactly one `DateTime64` column, expose its values in the `min_time` and `max_time` columns in `system.parts` and `system.parts_columns` tables. Add `min_time` and `max_time` columns to `system.parts_columns` table (these was inconsistency to the `system.parts` table). This closes [#18244](#). [#22011 \(alexey-milovidov\)](#).
- Supported `replication_alter_partitions_sync=1` setting in `clickhouse-copier` for moving partitions from helping table to destination. Decreased default timeouts. Fixes [#21911](#). [#21912 \(turbo jason\)](#).
- Show path to data directory of `EmbeddedRocksDB` tables in system tables. [#21903 \(tavplubix\)](#).
- Add profile event `HedgedRequestsChangeReplica`, change read data timeout from sec to ms. [#21886 \(Kruglov Pavel\)](#).
- DiskS3 (experimental feature under development). Fixed bug with the impossibility to move directory if the destination is not empty and cache disk is used. [#21837 \(Pavel Kovalenko\)](#).
- Better formatting for `Array` and `Map` data types in Web UI. [#21798 \(alexey-milovidov\)](#).
- Update clusters only if their configurations were updated. [#21685 \(Kruglov Pavel\)](#).

- Propagate query and session settings for distributed DDL queries. Set `distributed_ddl_entry_format_version` to 2 to enable this. Added `distributed_ddl_output_mode` setting. Supported modes: `none`, `throw` (default), `null_status_on_timeout` and `never_throw`. Miscellaneous fixes and improvements for Replicated database engine. #21535 (tavplubix).
- If `PODArray` was instantiated with element size that is neither a fraction or a multiple of 16, buffer overflow was possible. No bugs in current releases exist. #21533 (alexey-milovidov).
- Add `last_error_time`/`last_error_message`/`last_error_stacktrace`/`remote` columns for `system.errors`. #21529 (Azat Khuzhin).
- Add aliases `simpleJSONExtract`/`simpleJSONHas` to `visitParam`/`visitParamExtract{UInt, Int, Bool, Float, Raw, String}`. Fixes #21383. #21519 (fastio).
- Add setting `optimize_skip_unused_shards_limit` to limit the number of sharding key values for `optimize_skip_unused_shards`. #21512 (Azat Khuzhin).
- Improve `clickhouse-format` to not throw exception when there are extra spaces or comment after the last query, and throw exception early with readable message when format `ASTInsertQuery` with data . #21311 (flynn).
- Improve support of integer keys in data type `Map`. #21157 (Anton Popov).
- MaterializeMySQL: attempt to reconnect to MySQL if the connection is lost. #20961 (Håvard Kvålen).
- Support more cases to rewrite CROSS JOIN to INNER JOIN. #20392 (Vladimir).
- Do not create empty parts on INSERT when `optimize_on_insert` setting enabled. Fixes #20304. #20387 (Kruglov Pavel).
- MaterializeMySQL: add minmax skipping index for `_version` column. #20382 (Stig Bakken).
- Add option `--backslash` for `clickhouse-format`, which can add a backslash at the end of each line of the formatted query. #21494 (flynn).
- Now clickhouse will not throw `LOGICAL_ERROR` exception when we try to mutate the already covered part. Fixes #22013. #22291 (alesapin).

## Bug Fix

- Remove socket from epoll before cancelling packet receiver in `HedgedConnections` to prevent possible race. Fixes #22161. #22443 (Kruglov Pavel).
- Add (missing) memory accounting in parallel parsing routines. In previous versions OOM was possible when the resultset contains very large blocks of data. This closes #22008. #22425 (alexey-milovidov).
- Fix exception which may happen when `SELECT` has constant `WHERE` condition and source table has columns which names are digits. #22270 (LiuNeng).
- Fix query cancellation with `use_hedged_requests=0` and `async_socket_for_remote=1`. #22183 (Azat Khuzhin).
- Fix uncaught exception in `InterserverIOHTTPHandler`. #22146 (Azat Khuzhin).
- Fix docker entrypoint in case `http_port` is not in the config. #22132 (Ewout).
- Fix error `Invalid number of rows in Chunk` in `JOIN` with `TOTALS` and `arrayJoin`. Closes #19303. #22129 (Vladimir).
- Fix the background thread pool name which used to poll message from Kafka. The Kafka engine with the broken thread pool will not consume the message from message queue. #22122 (fastio).

- Fix waiting for `OPTIMIZE` and `ALTER` queries for `ReplicatedMergeTree` table engines. Now the query will not hang when the table was detached or restarted. #22118 (alesapin).
- Disable `async_socket_for_remote/use_hedged_requests` for buggy Linux kernels. #22109 (Azat Khuzhin).
- Docker entrypoint: avoid chown of `.` in case when `LOG_PATH` is empty. Closes #22100. #22102 (filimonov).
- The function `decrypt` was lacking a check for the minimal size of data encrypted in AEAD mode. This closes #21897. #22064 (alexey-milovidov).
- In rare case, merge for `CollapsingMergeTree` may create granule with `index_granularity + 1` rows. Because of this, internal check, added in #18928 (affects 21.2 and 21.3), may fail with error `Incomplete granules are not allowed while blocks are granules size`. This error did not allow parts to merge. #21976 (Nikolai Kochetov).
- Reverted #15454 that may cause significant increase in memory usage while loading external dictionaries of hashed type. This closes #21935. #21948 (Maksim Kita).
- Prevent hedged connections overlaps (`Unknown packet 9 from server error`). #21941 (Azat Khuzhin).
- Fix reading the HTTP POST request with "multipart/form-data" content type in some cases. #21936 (Ivan).
- Fix wrong `ORDER BY` results when a query contains window functions, and optimization for reading in primary key order is applied. Fixes #21828. #21915 (Alexander Kuzmenkov).
- Fix deadlock in first catboost model execution. Closes #13832. #21844 (Kruglov Pavel).
- Fix incorrect query result (and possible crash) which could happen when `WHERE` or `HAVING` condition is pushed before `GROUP BY`. Fixes #21773. #21841 (Nikolai Kochetov).
- Better error handling and logging in `WriteBufferFromS3`. #21836 (Pavel Kovalenko).
- Fix possible crashes in aggregate functions with combinator `Distinct`, while using two-level aggregation. This is a follow-up fix of #18365 . Can only reproduced in production env. #21818 (Amos Bird).
- Fix scalar subquery index analysis. This fixes #21717 , which was introduced in #18896. #21766 (Amos Bird).
- Fix bug for `ReplicatedMerge` table engines when `ALTER MODIFY COLUMN` query doesn't change the type of `Decimal` column if its size (32 bit or 64 bit) doesn't change. #21728 (alesapin).
- Fix possible infinite waiting when concurrent `OPTIMIZE` and `DROP` are run for `ReplicatedMergeTree`. #21716 (Azat Khuzhin).
- Fix function `arrayElement` with type `Map` for constant integer arguments. #21699 (Anton Popov).
- Fix SIGSEGV on not existing attributes from `ip_trie` with `access_to_key_from_attributes`. #21692 (Azat Khuzhin).
- Server now start accepting connections only after `DDLWorker` and dictionaries initialization. #21676 (Azat Khuzhin).
- Add type conversion for keys of tables of type `Join` (previously led to SIGSEGV). #21646 (Azat Khuzhin).
- Fix distributed requests cancellation (for example simple select from multiple shards with limit, i.e. `select * from remote('127.{2,3}', system.numbers) limit 100`) with `async_socket_for_remote=1`. #21643 (Azat Khuzhin).
- Fix `fsync_part_directory` for horizontal merge. #21642 (Azat Khuzhin).

- Remove unknown columns from joined table in `WHERE` for queries to external database engines (MySQL, PostgreSQL). close #14614, close #19288 (dup), close #19645 (dup). #21640 (Vladimir).
- `std::terminate` was called if there is an error writing data into s3. #21624 (Vladimir).
- Fix possible error `Cannot find column` when `optimize_skip_unused_shards` is enabled and zero shards are used. #21579 (Azat Khuzhin).
- In case if query has constant `WHERE` condition, and setting `optimize_skip_unused_shards` enabled, all shards may be skipped and query could return incorrect empty result. #21550 (Amos Bird).
- Fix table function `clusterAllReplicas` returns wrong `_shard_num`. close #21481. #21498 (flynn).
- Fix that S3 table holds old credentials after config update. #21457 (Grigory Pervakov).
- Fixed race on SSL object inside `SecureSocket` in Poco. #21456 (Nikita Mikhaylov).
- Fix Avro format parsing for Kafka. Fixes #21437. #21438 (Ilya Golshtein).
- Fix receive and send timeouts and non-blocking read in secure socket. #21429 (Kruglov Pavel).
- `force_drop_table` flag didn't work for MATERIALIZED VIEW, it's fixed. Fixes #18943. #20626 (tavplubix).
- Fix name clashes in `PredicateRewriteVisitor`. It caused incorrect WHERE filtration after full join. Close #20497. #20622 (Vladimir).

## Build/Testing/Packaging Improvement

- Add `Jepsen` tests for ClickHouse Keeper. #21677 (alesapin).
- Run stateless tests in parallel in CI. Depends on #22181. #22300 (alesapin).
- Enable status check for `SQLancer` CI run. #22015 (Ilya Yatsishin).
- Multiple preparations for PowerPC builds: Enable the bundled openldap on `ppc64le`. #22487 (Kfir Itzhak). Enable compiling on `ppc64le` with Clang. #22476 (Kfir Itzhak). Fix compiling boost on `ppc64le`. #22474 (Kfir Itzhak). Fix CMake error about internal CMake variable `CMAKE_ASM_COMPILE_OBJECT` not set on `ppc64le`. #22469 (Kfir Itzhak). Fix Fedora/RHEL/CentOS not finding `libclang_rt.builtins` on `ppc64le`. #22458 (Kfir Itzhak). Enable building with `jemalloc` on `ppc64le`. #22447 (Kfir Itzhak). Fix ClickHouse's config embedding and cctz's timezone embedding on `ppc64le`. #22445 (Kfir Itzhak). Fixed compiling on `ppc64le` and use the correct instruction pointer register on `ppc64le`. #22430 (Kfir Itzhak).
- Re-enable the S3 (AWS) library on `aarch64`. #22484 (Kfir Itzhak).
- Add `tzdata` to Docker containers because reading ORC formats requires it. This closes #14156. #22000 (alexey-milovidov).
- Introduce 2 arguments for `clickhouse-server` image Dockerfile: `deb_location` & `single_binary_location`. #21977 (filimonov).
- Allow to use clang-tidy with release builds by enabling assertions if it is used. #21914 (alexey-milovidov).
- Add `llvm-12` binaries name to search in cmake scripts. Implicit constants conversions to mute clang warnings. Updated submodules to build with CMake 3.19. Mute recursion in macro expansion in `readpassphrase` library. Deprecated `-fuse-ld` changed to `--ld-path` for clang. #21597 (Ilya Yatsishin).
- Updating `docker/test/testflows/runner/dockerd-entrypoint.sh` to use Yandex dockerhub-proxy, because Docker Hub has enabled very restrictive rate limits #21551 (vzakaznikov).
- Fix macOS shared lib build. #20184 (nvartolomei).

- Add `ctime` option to `zookeeper-dump-tree`. It allows to dump node creation time. #21842 (Ilya).

# ClickHouse release 21.3 (LTS)

## ClickHouse release v21.3, 2021-03-12

### Backward Incompatible Change

- Now it's not allowed to create MergeTree tables in old syntax with table TTL because it's just ignored. Attach of old tables is still possible. #20282 (alesapin).
- Now all case-insensitive function names will be rewritten to their canonical representations. This is needed for projection query routing (the upcoming feature). #20174 (Amos Bird).
- Fix creation of `TTL` in cases, when its expression is a function and it is the same as `ORDER BY` key. Now it's allowed to set custom aggregation to primary key columns in `TTL` with `GROUP BY`. Backward incompatible: For primary key columns, which are not in `GROUP BY` and aren't set explicitly now is applied function `any` instead of `max`, when `TTL` is expired. Also if you use `TTL` with `WHERE` or `GROUP BY` you can see exceptions at merges, while making rolling update. #15450 (Anton Popov).

### New Feature

- Add file engine settings: `engine_file_empty_if_not_exists` and `engine_file_truncate_on_insert`. #20620 (M0r64n).
- Add aggregate function `deltaSum` for summing the differences between consecutive rows. #20057 (Russ Frank).
- New `event_time_microseconds` column in `system.part_log` table. #20027 (Bharat Nallan).
- Added `timezoneOffset(datetime)` function which will give the offset from UTC in seconds. This close #issue:19850. #19962 (keenwolf).
- Add setting `insert_shard_id` to support insert data into specific shard from distributed table. #19961 (flynn).
- Function `reinterpretAs` updated to support big integers. Fixes #19691. #19858 (Maksim Kita).
- Added Server Side Encryption Customer Keys (the `x-amz-server-side-encryption-customer-(key/md5)` header) support in S3 client. See the link. Closes #19428. #19748 (Vladimir Chebotarev).
- Added `implicit_key` option for `executable` dictionary source. It allows to avoid printing key for every record if records comes in the same order as the input keys. Implements #14527. #19677 (Maksim Kita).
- Add quota type `query_selects` and `query_inserts`. #19603 (JackyWoo).
- Add function `extractTextFromHTML` #19600 (zlx19950903), (alexey-milovidov).
- Tables with `MergeTree*` engine now have two new table-level settings for query concurrency control. Setting `max_concurrent_queries` limits the number of concurrently executed queries which are related to this table. Setting `min_marks_to_honor_max_concurrent_queries` tells to apply previous setting only if query reads at least this number of marks. #19544 (Amos Bird).
- Added `file` function to read file from `user_files` directory as a String. This is different from the `file` table function. This implements #issue:18851. #19204 (keenwolf).

### Experimental feature

- Add experimental `Replicated` database engine. It replicates DDL queries across multiple hosts. #16193 (tavplubix).

- Introduce experimental support for window functions, enabled with `allow_experimental_window_functions = 1`. This is a preliminary, alpha-quality implementation that is not suitable for production use and will change in backward-incompatible ways in future releases. Please see [the documentation](#) for the list of supported features. [#20337 \(Alexander Kuzmenkov\)](#).
- Add the ability to backup/restore metadata files for DiskS3. [#18377 \(Pavel Kovalenko\)](#).

## Performance Improvement

- Hedged requests for remote queries. When setting `use_hedged_requests` enabled (off by default), allow to establish many connections with different replicas for query. New connection is enabled in case existent connection(s) with replica(s) were not established within `hedged_connection_timeout` or no data was received within `receive_data_timeout`. Query uses the first connection which send non empty progress packet (or data packet, if `allow_changing_replica_until_first_data_packet`); other connections are cancelled. Queries with `max_parallel_replicas > 1` are supported. [#19291 \(Kruglov Pavel\)](#). This allows to significantly reduce tail latencies on very large clusters.
- Added support for `PREWHERE` (and enable the corresponding optimization) when tables have row-level security expressions specified. [#19576 \(Denis Glazachev\)](#).
- The setting `distributed_aggregation_memory_efficient` is enabled by default. It will lower memory usage and improve performance of distributed queries. [#20599 \(alexey-milovidov\)](#).
- Improve performance of GROUP BY multiple fixed size keys. [#20472 \(alexey-milovidov\)](#).
- Improve performance of aggregate functions by more strict aliasing. [#19946 \(alexey-milovidov\)](#).
- Speed up reading from `Memory` tables in extreme cases (when reading speed is in order of 50 GB/sec) by simplification of pipeline and (consequently) less lock contention in pipeline scheduling. [#20468 \(alexey-milovidov\)](#).
- Partially reimplement HTTP server to make it making less copies of incoming and outgoing data. It gives up to 1.5 performance improvement on inserting long records over HTTP. [#19516 \(Ivan\)](#).
- Add `compress` setting for `Memory` tables. If it's enabled the table will use less RAM. On some machines and datasets it can also work faster on `SELECT`, but it is not always the case. This closes [#20093](#). Note: there are reasons why `Memory` tables can work slower than `MergeTree`: (1) lack of compression (2) static size of blocks (3) lack of indices and prewhere... [#20168 \(alexey-milovidov\)](#).
- Slightly better code in aggregation. [#20978 \(alexey-milovidov\)](#).
- Add back `intDiv/modulo` specializations for better performance. This fixes [#21293](#). The regression was introduced in <https://github.com/ClickHouse/ClickHouse/pull/18145>. [#21307 \(Amos Bird\)](#).
- Do not squash blocks too much on `INSERT SELECT` if inserting into `Memory` table. In previous versions inefficient data representation was created in `Memory` table after `INSERT SELECT`. This closes [#13052](#). [#20169 \(alexey-milovidov\)](#).
- Fix at least one case when `DataType` parser may have exponential complexity (found by fuzzer). This closes [#20096](#). [#20132 \(alexey-milovidov\)](#).
- Parallelize `SELECT` with `FINAL` for single part with `level > 0` when `do_not_merge_across_partitions_select_final` setting is 1. [#19375 \(Kruglov Pavel\)](#).
- Fill only requested columns when querying `system.parts` and `system.parts_columns`. Closes [#19570](#). [#21035 \(Anmol Arora\)](#).
- Perform algebraic optimizations of arithmetic expressions inside `avg` aggregate function. close [#20092](#). [#20183 \(flynn\)](#).

## Improvement

- Case-insensitive compression methods for table functions. Also fixed LZMA compression method which was checked in upper case. [#21416](#) ([Vladimir Chebotarev](#)).
- Add two settings to delay or throw error during insertion when there are too many inactive parts. This is useful when server fails to clean up parts quickly enough. [#20178](#) ([Amos Bird](#)).
- Provide better compatibility for mysql clients. 1. mysql jdbc 2. mycli. [#21367](#) ([Amos Bird](#)).
- Forbid to drop a column if it's referenced by materialized view. Closes [#21164](#). [#21303](#) ([flynn](#)).
- MySQL dictionary source will now retry unexpected connection failures (Lost connection to MySQL server during query) which sometimes happen on SSL/TLS connections. [#21237](#) ([Alexander Kazakov](#)).
- Usability improvement: more consistent `DateTime64` parsing: recognize the case when unix timestamp with subsecond resolution is specified as scaled integer (like `1111111111222` instead of `1111111111.222`). This closes [#13194](#). [#21053](#) ([alexey-milovidov](#)).
- Do only merging of sorted blocks on initiator with `distributed_group_by_no_merge`. [#20882](#) ([Azat Khuzhin](#)).
- When loading config for mysql source ClickHouse will now randomize the list of replicas with the same priority to ensure the round-robin logic of picking mysql endpoint. This closes [#20629](#). [#20632](#) ([Alexander Kazakov](#)).
- Function 'reinterpretAs(x, Type)' renamed into 'reinterpret(x, Type)'. [#20611](#) ([Maksim Kita](#)).
- Support vhost for RabbitMQ engine [#20576](#). [#20596](#) ([Ksenia Sumarokova](#)).
- Improved serialization for data types combined of Arrays and Tuples. Improved matching enum data types to protobuf enum type. Fixed serialization of the `Map` data type. Omitted values are now set by default. [#20506](#) ([Vitaly Baranov](#)).
- Fixed race between execution of distributed DDL tasks and cleanup of DDL queue. Now DDL task cannot be removed from ZooKeeper if there are active workers. Fixes [#20016](#). [#20448](#) ([tavplubix](#)).
- Make FQDN and other DNS related functions work correctly in alpine images. [#20336](#) ([filimonov](#)).
- Do not allow early constant folding of explicitly forbidden functions. [#20303](#) ([Azat Khuzhin](#)).
- Implicit conversion from integer to Decimal type might succeed if integer value does not fit into Decimal type. Now it throws `ARGUMENT_OUT_OF_BOUND`. [#20232](#) ([tavplubix](#)).
- Lockless `SYSTEM FLUSH DISTRIBUTED`. [#20215](#) ([Azat Khuzhin](#)).
- Normalize `count(constant)`, `sum(1)` to `count()`. This is needed for projection query routing. [#20175](#) ([Amos Bird](#)).
- Support all native integer types in bitmap functions. [#20171](#) ([Amos Bird](#)).
- Updated `CacheDictionary`, `ComplexCacheDictionary`, `SSDCacheDictionary`, `SSDComplexKeyDictionary` to use `LRUHashMap` as underlying index. [#20164](#) ([Maksim Kita](#)).
- The setting `access_management` is now configurable on startup by providing `CLICKHOUSE_DEFAULT_ACCESS_MANAGEMENT`, defaults to disabled (0) which was the prior value. [#20139](#) ([Marquitos](#)).
- Fix `toDateTime64(toDate()/toDateTime())` for `DateTime64` - Implement `DateTime64` clamping to match `DateTime` behaviour. [#20131](#) ([Azat Khuzhin](#)).

- Quota improvements: SHOW TABLES is now considered as one query in the quota calculations, not two queries. SYSTEM queries now consume quota. Fix calculation of interval's end in quota consumption. [#20106](#) ([Vitaly Baranov](#)).
- Supports path IN (set) expressions for system.zookeeper table. [#20105](#) ([小路](#)).
- Show full details of MaterializeMySQL tables in system.tables. [#20051](#) ([Stig Bakken](#)).
- Fix data race in executable dictionary that was possible only on misuse (when the script returns data ignoring its input). [#20045](#) ([alexey-milovidov](#)).
- The value of MYSQL\_OPT\_RECONNECT option can now be controlled by "opt\_reconnect" parameter in the config section of mysql replica. [#19998](#) ([Alexander Kazakov](#)).
- If user calls JSONExtract function with Float32 type requested, allow inaccurate conversion to the result type. For example the number 0.1 in JSON is double precision and is not representable in Float32, but the user still wants to get it. Previous versions return 0 for non-Nullable type and NULL for Nullable type to indicate that conversion is imprecise. The logic was 100% correct but it was surprising to users and leading to questions. This closes [#13962](#). [#19960](#) ([alexey-milovidov](#)).
- Add conversion of block structure for INSERT into Distributed tables if it does not match. [#19947](#) ([Azat Khuzhin](#)).
- Improvement for the system.distributed\_ddl\_queue table. Initialize MaxDDLEntryID to the last value after restarting. Before this PR, MaxDDLEntryID will remain zero until a new DDLTask is processed. [#19924](#) ([Amos Bird](#)).
- Show MaterializeMySQL tables in system.parts. [#19770](#) ([Stig Bakken](#)).
- Add separate config directive for Buffer profile. [#19721](#) ([Azat Khuzhin](#)).
- Move conditions that are not related to JOIN to WHERE clause. [#18720](#). [#19685](#) ([hexiaoting](#)).
- Add ability to throttle INSERT into Distributed based on amount of pending bytes for async send (bytes\_to\_delay\_insert/max\_delay\_to\_insert and bytes\_to\_throw\_insert settings for Distributed engine has been added). [#19673](#) ([Azat Khuzhin](#)).
- Fix some rare cases when write errors can be ignored in destructors. [#19451](#) ([Azat Khuzhin](#)).
- Print inline frames in stack traces for fatal errors. [#19317](#) ([Ivan](#)).

## Bug Fix

- Fix redundant reconnects to ZooKeeper and the possibility of two active sessions for a single clickhouse server. Both problems introduced in [#14678](#). [#21264](#) ([alesapin](#)).
- Fix error Bad cast from type ... to DB::ColumnLowCardinality while inserting into table with LowCardinality column from Values format. Fixes [#21140](#) [#21357](#) ([Nikolai Kochetov](#)).
- Fix a deadlock in ALTER DELETE mutations for non replicated MergeTree table engines when the predicate contains the table itself. Fixes [#20558](#). [#21477](#) ([alesapin](#)).
- Fix SIGSEGV for distributed queries on failures. [#21434](#) ([Azat Khuzhin](#)).
- Now ALTER MODIFY COLUMN queries will correctly affect changes in partition key, skip indices, TTLs, and so on. Fixes [#13675](#). [#21334](#) ([alesapin](#)).
- Fix bug with join\_use\_nulls and joining TOTALS from subqueries. This closes [#19362](#) and [#21137](#). [#21248](#) ([vdimir](#)).
- Fix crash in EXPLAIN for query with UNION. Fixes [#20876](#), [#21170](#). [#21246](#) ([flynn](#)).

- Now mutations allowed only for table engines that support them (MergeTree family, Memory, MaterializedView). Other engines will report a more clear error. Fixes #21168. #21183 (alesapin).
- Fixes #21112. Fixed bug that could cause duplicates with insert query (if one of the callbacks came a little too late). #21138 (Kseniia Sumarokova).
- Fix `input_format_null_as_default` take effective when types are nullable. This fixes #21116 . #21121 (Amos Bird).
- fix bug related to cast Tuple to Map. Closes #21029. #21120 (hexiaoting).
- Fix the metadata leak when the Replicated\*MergeTree with custom (non default) ZooKeeper cluster is dropped. #21119 (fastio).
- Fix type mismatch issue when using LowCardinality keys in joinGet. This fixes #21114. #21117 (Amos Bird).
- fix `default_replica_path` and `default_replica_name` values are useless on Replicated(\*)MergeTree engine when the engine needs specify other parameters. #21060 (mxzlxy).
- Out of bound memory access was possible when formatting specifically crafted out of range value of type `DateTime64`. This closes #20494. This closes #20543. #21023 (alexey-milovidov).
- Block parallel insertions into storage join. #21009 (vdimir).
- Fixed behaviour, when `ALTER MODIFY COLUMN` created mutation, that will knowingly fail. #21007 (Anton Popov).
- Closes #9969. Fixed Brotli http compression error, which reproduced for large data sizes, slightly complicated structure and with json output format. Update Brotli to the latest version to include the "fix rare access to uninitialized data in ring-buffer". #20991 (Kseniia Sumarokova).
- Fix 'Empty task was returned from async task queue' on query cancellation. #20881 (Azat Khuzhin).
- USE database; query did not work when using MySQL 5.7 client to connect to ClickHouse server, it's fixed. Fixes #18926. #20878 (tavplubix).
- Fix usage of `-Distinct` combinator with `-State` combinator in aggregate functions. #20866 (Anton Popov).
- Fix subquery with union distinct and limit clause. close #20597. #20610 (flynn).
- Fixed inconsistent behavior of dictionary in case of queries where we look for absent keys in dictionary. #20578 (Nikita Mikhaylov).
- Fix the number of threads for scalar subqueries and subqueries for index (after #19007 single thread was always used). Fixes #20457, #20512. #20550 (Nikolai Kochetov).
- Fix crash which could happen if unknown packet was received from remove query (was introduced in #17868). #20547 (Azat Khuzhin).
- Add proper checks while parsing directory names for async INSERT (fixes SIGSEGV). #20498 (Azat Khuzhin).
- Fix function `transform` does not work properly for floating point keys. Closes #20460. #20479 (flynn).
- Fix infinite loop when propagating WITH aliases to subqueries. This fixes #20388. #20476 (Amos Bird).
- Fix abnormal server termination when http client goes away. #20464 (Azat Khuzhin).
- Fix `LOGICAL_ERROR` for `join_use_nulls=1` when JOIN contains const from SELECT. #20461 (Azat Khuzhin).

- Check if table function `view` is used in expression list and throw an error. This fixes #20342. #20350 ([Amos Bird](#)).
- Avoid invalid dereference in `RANGE_HASHED()` dictionary. #20345 ([Azat Khuzhin](#)).
- Fix null dereference with `join_use_nulls=1`. #20344 ([Azat Khuzhin](#)).
- Fix incorrect result of binary operations between two constant decimals of different scale. Fixes #20283. #20339 ([Maksim Kita](#)).
- Fix too often retries of failed background tasks for `ReplicatedMergeTree` table engines family. This could lead to too verbose logging and increased CPU load. Fixes #20203. #20335 ([alesapin](#)).
- Restrict to `DROP` or `RENAME` version column of `*CollapsingMergeTree` and `ReplacingMergeTree` table engines. #20300 ([alesapin](#)).
- Fixed the behavior when in case of broken JSON we tried to read the whole file into memory which leads to exception from the allocator. Fixes #19719. #20286 ([Nikita Mikhaylov](#)).
- Fix exception during vertical merge for `MergeTree` table engines family which don't allow to perform vertical merges. Fixes #20259. #20279 ([alesapin](#)).
- Fix rare server crash on config reload during the shutdown. Fixes #19689. #20224 ([alesapin](#)).
- Fix CTE when using in `INSERT SELECT`. This fixes #20187, fixes #20195. #20211 ([Amos Bird](#)).
- Fixes #19314. #20156 ([Ivan](#)).
- fix `toMinute` function to handle special timezone correctly. #20149 ([keenwolf](#)).
- Fix server crash after query with `if` function with `Tuple` type of then/else branches result. `Tuple` type must contain `Array` or another complex type. Fixes #18356. #20133 ([alesapin](#)).
- The `MongoDB` table engine now establishes connection only when it's going to read data. `ATTACH TABLE` won't try to connect anymore. #20110 ([Vitaly Baranov](#)).
- Bugfix in `StorageJoin`. #20079 ([vdimir](#)).
- Fix the case when calculating modulo of division of negative number by small divisor, the resulting data type was not large enough to accomodate the negative result. This closes #20052. #20067 ([alexey-milovidov](#)).
- MaterializeMySQL: Fix replication for statements that update several tables. #20066 ([Håvard Kvålen](#)).
- Prevent "Connection refused" in docker during initialization script execution. #20012 ([filimonov](#)).
- `EmbeddedRocksDB` is an experimental storage. Fix the issue with lack of proper type checking. Simplified code. This closes #19967. #19972 ([alexey-milovidov](#)).
- Fix a segfault in function `fromModifiedJulianDay` when the argument type is `Nullable(T)` for any integral types other than `Int32`. #19959 ([PHO](#)).
- BloomFilter index crash fix. Fixes #19757. #19884 ([Maksim Kita](#)).
- Deadlock was possible if `system.text_log` is enabled. This fixes #19874. #19875 ([alexey-milovidov](#)).
- Fix starting the server with tables having default expressions containing `dictGet()`. Allow getting return type of `dictGet()` without loading dictionary. #19805 ([Vitaly Baranov](#)).
- Fix clickhouse-client abort exception while executing only `select`. #19790 ([taiyang-li](#)).

- Fix a bug that moving pieces to destination table may failed in case of launching multiple clickhouse-copiers. [#19743](#) ([madianjun](#)).
- Background thread which executes `ON CLUSTER` queries might hang waiting for dropped replicated table to do something. It's fixed. [#19684](#) ([yiguolei](#)).

## Build/Testing/Packaging Improvement

- Allow to build ClickHouse with AVX-2 enabled globally. It gives slight performance benefits on modern CPUs. Not recommended for production and will not be supported as official build for now. [#20180](#) ([alexey-milovidov](#)).
- Fix some of the issues found by Coverity. See [#19964](#). [#20010](#) ([alexey-milovidov](#)).
- Allow to start up with modified binary under gdb. In previous version if you set up breakpoint in gdb before start, server will refuse to start up due to failed integrity check. [#21258](#) ([alexey-milovidov](#)).
- Add a test for different compression methods in Kafka. [#21111](#) ([filimonov](#)).
- Fixed port clash from `test_storage_kerberized_hdfs` test. [#19974](#) ([Ilya Yatsishin](#)).
- Print `stdout` and `stderr` to log when failed to start docker in integration tests. Before this PR there was a very short error message in this case which didn't help to investigate the problems. [#20631](#) ([Vitaly Baranov](#)).

## ClickHouse release 21.2

### ClickHouse release v21.2.2.8-stable, 2021-02-07

#### Backward Incompatible Change

- Bitwise functions (`bitAnd`, `bitOr`, etc) are forbidden for floating point arguments. Now you have to do explicit cast to integer. [#19853](#) ([Azat Khuzhin](#)).
- Forbid `lcm/gcd` for floats. [#19532](#) ([Azat Khuzhin](#)).
- Fix memory tracking for `OPTIMIZE TABLE/merges`; account query memory limits and sampling for `OPTIMIZE TABLE/merges`. [#18772](#) ([Azat Khuzhin](#)).
- Disallow floating point column as partition key, see [#18421](#). [#18464](#) ([hexiaoting](#)).
- Excessive parenthesis in type definitions no longer supported, example: `Array((UInt8))`.

#### New Feature

- Added `PostgreSQL` table engine (both select/insert, with support for multidimensional arrays), also as table function. Added `PostgreSQL` dictionary source. Added `PostgreSQL` database engine. [#18554](#) ([Kseniia Sumarokova](#)).
- Data type `Nested` now supports arbitrary levels of nesting. Introduced subcolumns of complex types, such as `size0` in `Array`, `null` in `Nullable`, names of `Tuple` elements, which can be read without reading of whole column. [#17310](#) ([Anton Popov](#)).
- Added `Nullable` support for `FlatDictionary`, `HashedDictionary`, `ComplexKeyHashedDictionary`, `DirectDictionary`, `ComplexKeyDirectDictionary`, `RangeHashedDictionary`. [#18236](#) ([Maksim Kita](#)).
- Adds a new table called `system.distributed_ddl_queue` that displays the queries in the DDL worker queue. [#17656](#) ([Bharat Nallan](#)).

- Added support of mapping LDAP group names, and attribute values in general, to local roles for users from Ildap user directories. #17211 (Denis Glazachev).
- Support insert into table function `cluster`, and for both table functions `remote` and `cluster`, support distributing data across nodes by specify sharding key. Close #16752. #18264 (flynn).
- Add function `decodeXMLComponent` to decode characters for XML. Example: `SELECT decodeXMLComponent('Hello,"world"!')` #17659. #18542 (nauta).
- Added functions `parseDateTimeBestEffortUSOrZero`, `parseDateTimeBestEffortUSOrNull`. #19712 (Maksim Kita).
- Add `sign` math function. #19527 (flynn).
- Add information about used features (functions, table engines, etc) into `system.query_log`. #18495. #19371 (Ksenia Sumarokova).
- Function `formatDateTime` support the `%Q` modification to format date to quarter. #19224 (Jianmei Zhang).
- Support MetaKey+Enter hotkey binding in play UI. #19012 (sundyli).
- Add three functions for map data type: 1. `mapContains(map, key)` to check weather map.keys include the second parameter key. 2. `mapKeys(map)` return all the keys in Array format 3. `mapValues(map)` return all the values in Array format. #18788 (hexiaoting).
- Add `log_comment` setting related to #18494. #18549 (Zijie Lu).
- Add support of tuple argument to `argMin` and `argMax` functions. #17359 (Ildus Kurbangaliev).
- Support `EXISTS VIEW` syntax. #18552 (Du Chuan).
- Add `SELECT ALL` syntax. closes #18706. #18723 (flynn).

## Performance Improvement

- Faster parts removal by lowering the number of `stat` syscalls. This returns the optimization that existed while ago. More safe interface of `IDisk`. This closes #19065. #19086 (alexey-milovidov).
- Aliases declared in `WITH` statement are properly used in index analysis. Queries like `WITH column AS alias SELECT ... WHERE alias = ...` may use index now. #18896 (Amos Bird).
- Add `optimize_alias_column_prediction` (on by default), that will: - Respect aliased columns in `WHERE` during partition pruning and skipping data using secondary indexes; - Respect aliased columns in `WHERE` for trivial count queries for `optimize_trivial_count`; - Respect aliased columns in `GROUP BY/ORDER BY` for `optimize_aggregation_in_order/optimize_read_in_order`. #16995 (sundyli).
- Speed up aggregate function `sum`. Improvement only visible on synthetic benchmarks and not very practical. #19216 (alexey-milovidov).
- Update `libc++` and use another ABI to provide better performance. #18914 (Danila Kutenin).
- Rewrite `sumIf()` and `sum(if())` function to `countIf()` function when logically equivalent. #17041 (flynn).
- Use a connection pool for S3 connections, controlled by the `s3_max_connections` settings. #13405 (Vladimir Chebotarev).
- Add support for zstd long option for better compression of string columns to save space. #17184 (ygrek).
- Slightly improve server latency by removing access to configuration on every connection. #19863 (alexey-milovidov).
- Reduce lock contention for multiple layers of the `Buffer` engine. #19379 (Azat Khuzhin).

- Support splitting `Filter` step of query plan into `Expression + Filter` pair. Together with `Expression + Expression` merging optimization (#17458) it may delay execution for some expressions after `Filter` step. #19253 (Nikolai Kochetov).

## Improvement

- `SELECT count()` FROM table now can be executed if only one any column can be selected from the table. This PR fixes #10639. #18233 (Vitaly Baranov).
- Set charset to `utf8mb4` when interacting with remote MySQL servers. Fixes #19795. #19800 (alexey-milovidov).
- `S3` table function now supports `auto` compression mode (autodetect). This closes #18754. #19793 (Vladimir Chebotarev).
- Correctly output infinite arguments for `formatReadableTimeDelta` function. In previous versions, there was implicit conversion to implementation specific integer value. #19791 (alexey-milovidov).
- Table function `S3` will use global region if the region can't be determined exactly. This closes #10998. #19750 (Vladimir Chebotarev).
- In distributed queries if the setting `async_socket_for_remote` is enabled, it was possible to get stack overflow at least in debug build configuration if very deeply nested data type is used in table (e.g. `Array(Array(Array(...more...))))`). This fixes #19108. This change introduces minor backward incompatibility: excessive parenthesis in type definitions no longer supported, example: `Array((UInt8))`. #19736 (alexey-milovidov).
- Add separate pool for message brokers (RabbitMQ and Kafka). #19722 (Azat Khuzhin).
- Fix rare `max_number_of_merges_with_ttl_in_pool` limit overrun (more merges with TTL can be assigned) for non-replicated MergeTree. #19708 (alesapin).
- Dictionary: better error message during attribute parsing. #19678 (Maksim Kita).
- Add an option to disable validation of checksums on reading. Should never be used in production. Please do not expect any benefits in disabling it. It may only be used for experiments and benchmarks. The setting only applicable for tables of MergeTree family. Checksums are always validated for other table engines and when receiving data over network. In my observations there is no performance difference or it is less than 0.5%. #19588 (alexey-milovidov).
- Support constant result in function `multilf`. #19533 (Maksim Kita).
- Enable function `length/empty/notEmpty` for datatype Map, which returns keys number in Map. #19530 (taiyang-li).
- Add `--reconnect` option to `clickhouse-benchmark`. When this option is specified, it will reconnect before every request. This is needed for testing. #19872 (alexey-milovidov).
- Support using the new location of `.debug` file. This fixes #19348. #19520 (Amos Bird).
- `toIPv6` function parses `IPv4` addresses. #19518 (Bharat Nallan).
- Add `http_referer` field to `system.query_log`, `system.processes`, etc. This closes #19389. #19390 (alexey-milovidov).
- Improve MySQL compatibility by making more functions case insensitive and adding aliases. #19387 (Daniil Kondratyev).
- Add metrics for MergeTree parts (Wide/Compact/InMemory) types. #19381 (Azat Khuzhin).
- Allow docker to be executed with arbitrary uid. #19374 (filimonov).

- Fix wrong alignment of values of `IPv4` data type in Pretty formats. They were aligned to the right, not to the left. This closes [#19184](#). [#19339](#) ([alexey-milovidov](#)).
- Allow change `max_server_memory_usage` without restart. This closes [#18154](#). [#19186](#) ([alexey-milovidov](#)).
- The exception when function `bar` is called with certain NaN argument may be slightly misleading in previous versions. This fixes [#19088](#). [#19107](#) ([alexey-milovidov](#)).
- Explicitly set uid / gid of clickhouse user & group to the fixed values (101) in clickhouse-server images. [#19096](#) ([filimonov](#)).
- Fixed `PeekableReadBuffer`: Memory limit exceed error when inserting data with huge strings. Fixes [#18690](#). [#18979](#) ([tavplubix](#)).
- Docker image: several improvements for clickhouse-server entrypoint. [#18954](#) ([filimonov](#)).
- Add `normalizeQueryKeepNames` and `normalizedQueryHashKeepNames` to normalize queries without masking long names with ?. This helps better analyze complex query logs. [#18910](#) ([Amos Bird](#)).
- Check per-block checksum of the distributed batch on the sender before sending (without reading the file twice, the checksums will be verified while reading), this will avoid stuck of the INSERT on the receiver (on truncated .bin file on the sender). Avoid reading .bin files twice for batched INSERT (it was required to calculate rows/bytes to take squashing into account, now this information included into the header, backward compatible is preserved). [#18853](#) ([Azat Khuzhin](#)).
- Fix issues with RIGHT and FULL JOIN of tables with aggregate function states. In previous versions exception about `cloneResized` method was thrown. [#18818](#) ([templarzq](#)).
- Added prefix-based S3 endpoint settings. [#18812](#) ([Vladimir Chebotarev](#)).
- Add [`UInt8`, `UInt16`, `UInt32`, `UInt64`] arguments types support for `bitmapTransform`, `bitmapSubsetInRange`, `bitmapSubsetLimit`, `bitmapContains` functions. This closes [#18713](#). [#18791](#) ([sundyli](#)).
- Allow CTE (Common Table Expressions) to be further aliased. Propagate CSE (Common Subexpressions Elimination) to subqueries in the same level when `enable_global_with_statement = 1`. This fixes [#17378](#). This fixes <https://github.com/ClickHouse/ClickHouse/pull/16575#issuecomment-753416235>. [#18684](#) ([Amos Bird](#)).
- Update librdkafka to v1.6.0-RC2. Fixes [#18668](#). [#18671](#) ([filimonov](#)).
- In case of unexpected exceptions automatically restart background thread which is responsible for execution of distributed DDL queries. Fixes [#17991](#). [#18285](#) ([徐忻](#)).
- Updated AWS C++ SDK in order to utilize global regions in S3. [#17870](#) ([Vladimir Chebotarev](#)).
- Added support for `WITH ... [AND] [PERIODIC] REFRESH [interval_in_sec]` clause when creating LIVE VIEW tables. [#14822](#) ([vzakaznikov](#)).
- Restrict `MODIFY TTL` queries for `MergeTree` tables created in old syntax. Previously the query succeeded, but actually it had no effect. [#19064](#) ([Anton Popov](#)).

## Bug Fix

- Fix index analysis of binary functions with constant argument which leads to wrong query results. This fixes [#18364](#). [#18373](#) ([Amos Bird](#)).
- Fix starting the server with tables having default expressions containing `dictGet()`. Allow getting return type of `dictGet()` without loading dictionary. [#19805](#) ([Vitaly Baranov](#)).

- Fix server crash after query with `if` function with `Tuple` type of then/else branches result. `Tuple` type must contain `Array` or another complex type. Fixes #18356. #20133 (alesapin).
- MaterializeMySQL (experimental feature): Fix replication for statements that update several tables. #20066 (Håvard Kvålen).
- Prevent "Connection refused" in docker during initialization script execution. #20012 (filimonov).
- EmbeddedRocksDB is an experimental storage. Fix the issue with lack of proper type checking. Simplified code. This closes #19967. #19972 (alexey-milovidov).
- Fix a segfault in function `fromModifiedJulianDay` when the argument type is `Nullable(T)` for any integral types other than `Int32`. #19959 (PHO).
- The function `greatCircleAngle` returned inaccurate results in previous versions. This closes #19769. #19789 (alexey-milovidov).
- Fix rare bug when some replicated operations (like mutation) cannot process some parts after data corruption. Fixes #19593. #19702 (alesapin).
- Background thread which executes `ON CLUSTER` queries might hang waiting for dropped replicated table to do something. It's fixed. #19684 (yiguolei).
- Fix wrong deserialization of columns description. It makes `INSERT` into a table with a column named \ impossible. #19479 (alexey-milovidov).
- Mark distributed batch as broken in case of empty data block in one of files. #19449 (Azat Khuzhin).
- Fixed very rare bug that might cause mutation to hang after `DROP/DETACH/REPLACE/MOVE PARTITION`. It was partially fixed by #15537 for the most cases. #19443 (tavplubix).
- Fix possible error `Extremes` transform was already added to pipeline Fixes #14100. #19430 (Nikolai Kochetov).
- Fix default value in join types with non-zero default (e.g. some Enums). Closes #18197. #19360 (vdimir).
- Do not mark file for distributed send as broken on EOF. #19290 (Azat Khuzhin).
- Fix leaking of pipe fd for `async_socket_for_remote`. #19153 (Azat Khuzhin).
- Fix infinite reading from file in `ORC` format (was introduced in #10580). Fixes #19095. #19134 (Nikolai Kochetov).
- Fix issue in merge tree data writer which can lead to marks with bigger size than fixed granularity size. Fixes #18913. #19123 (alesapin).
- Fix startup bug when clickhouse was not able to read compression codec from `LowCardinality(Nullable(...))` and throws exception `Attempt to read after EOF`. Fixes #18340. #19101 (alesapin).
- Simplify the implementation of `tupleHammingDistance`. Support for tuples of any equal length. Fixes #19029. #19084 (Nikolai Kochetov).
- Make sure `groupUniqArray` returns correct type for argument of `Enum` type. This closes #17875. #19019 (alexey-milovidov).
- Fix possible error `Expected single dictionary argument for function` if use function `ignore` with `LowCardinality` argument. Fixes #14275. #19016 (Nikolai Kochetov).
- Fix inserting of `LowCardinality` column to table with `TinyLog` engine. Fixes #18629. #19010 (Nikolai Kochetov).

- Fix minor issue in JOIN: Join tries to materialize const columns, but our code waits for them in other places. [#18982](#) ([Nikita Mikhaylov](#)).
- Disable `optimize_move_functions_out_of_any` because optimization is not always correct. This closes [#18051](#). This closes [#18973](#). [#18981](#) ([alexey-milovidov](#)).
- Fix possible exception `QueryPipeline` stream: different number of columns caused by merging of query plan's Expression steps. Fixes [#18190](#). [#18980](#) ([Nikolai Kochetov](#)).
- Fixed very rare deadlock at shutdown. [#18977](#) ([tavplubix](#)).
- Fixed rare crashes when server run out of memory. [#18976](#) ([tavplubix](#)).
- Fix incorrect behavior when `ALTER TABLE ... DROP PART 'part_name'` query removes all deduplication blocks for the whole partition. Fixes [#18874](#). [#18969](#) ([alesapin](#)).
- Fixed issue [#18894](#) Add a check to avoid exception when long column alias('table.column' style, usually auto-generated by BI tools like Looker) equals to long table name. [#18968](#) ([Daniel Qin](#)).
- Fix error `Task was not found in task queue` (possible only for remote queries, with `async_socket_for_remote = 1`). [#18964](#) ([Nikolai Kochetov](#)).
- Fix bug when mutation with some escaped text (like `ALTER ... UPDATE e = CAST('foo', 'Enum8(\\'foo\' = 1')`) serialized incorrectly. Fixes [#18878](#). [#18944](#) ([alesapin](#)).
- ATTACH PARTITION will reset mutations. [#18804](#). [#18935](#) ([fastio](#)).
- Fix issue with `bitmapOrCardinality` that may lead to nullptr dereference. This closes [#18911](#). [#18912](#) ([sundyli](#)).
- Fixed Attempt to read after eof error when trying to `CAST NULL` from `Nullable(String)` to `Nullable(Decimal(P, S))`. Now function `CAST` returns `NULL` when it cannot parse decimal from nullable string. Fixes [#7690](#). [#18718](#) ([Winter Zhang](#)).
- Fix data type convert issue for MySQL engine. [#18124](#) ([bo zeng](#)).
- Fix clickhouse-client abort exception while executing only `select`. [#19790](#) ([taiyang-li](#)).

## Build/Testing/Packaging Improvement

- Run `SQLancer` (logical SQL fuzzer) in CI. [#19006](#) ([Ilya Yatsishin](#)).
- Query Fuzzer will fuzz newly added tests more extensively. This closes [#18916](#). [#19185](#) ([alexey-milovidov](#)).
- Integrate with `Big List of Naughty Strings` for better fuzzing. [#19480](#) ([alexey-milovidov](#)).
- Add integration tests run with MSan. [#18974](#) ([alesapin](#)).
- Fixed MemorySanitizer errors in cyrus-sasl and musl. [#19821](#) ([Ilya Yatsishin](#)).
- Insufficient arguments check in `positionCaseInsensitiveUTF8` function triggered address sanitizer. [#19720](#) ([alexey-milovidov](#)).
- Remove --project-directory for docker-compose in integration test. Fix logs formatting from docker container. [#19706](#) ([Ilya Yatsishin](#)).
- Made generation of macros.xml easier for integration tests. No more excessive logging from dicttoxml. dicttoxml project is not active for 5+ years. [#19697](#) ([Ilya Yatsishin](#)).
- Allow to explicitly enable or disable watchdog via environment variable `CLICKHOUSE_WATCHDOG_ENABLE`. By default it is enabled if server is not attached to terminal. [#19522](#) ([alexey-milovidov](#)).

- Allow building ClickHouse with Kafka support on arm64. #19369 (filimonov).
- Allow building librdkafka without ssl. #19337 (filimonov).
- Restore Kafka input in FreeBSD builds. #18924 (Alexandre Snarskii).
- Fix potential nullptr dereference in table function VALUES. #19357 (alexey-milovidov).
- Avoid UBSan reports in arrayElement function, substring and arraySum. Fixes #19305. Fixes #19287. This closes #19336. #19347 (alexey-milovidov).

## ClickHouse release 21.1

### ClickHouse release v21.1.3.32-stable, 2021-02-03

#### Bug Fix

- BloomFilter index crash fix. Fixes #19757. #19884 (Maksim Kita).
- Fix crash when pushing down predicates to union distinct subquery. This fixes #19855. #19861 (Amos Bird).
- Fix filtering by UInt8 greater than 127. #19799 (Anton Popov).
- In previous versions, unusual arguments for function arrayEnumerateUniq may cause crash or infinite loop. This closes #19787. #19788 (alexey-milovidov).
- Fixed stack overflow when using accurate comparison of arithmetic type with string type. #19773 (tavplubix).
- Fix crash when nested column name was used in WHERE or PREWHERE. Fixes #19755. #19763 (Nikolai Kochetov).
- Fix a segmentation fault in bitmapAndnot function. Fixes #19668. #19713 (Maksim Kita).
- Some functions with big integers may cause segfault. Big integers is experimental feature. This closes #19667. #19672 (alexey-milovidov).
- Fix wrong result of function neighbor for LowCardinality argument. Fixes #10333. #19617 (Nikolai Kochetov).
- Fix use-after-free of the CompressedWriteBuffer in Connection after disconnect. #19599 (Azat Khuzhin).
- DROP/DETACH TABLE table ON CLUSTER cluster SYNC query might hang, it's fixed. Fixes #19568. #19572 (tavplubix).
- Query CREATE DICTIONARY id expression fix. #19571 (Maksim Kita).
- Fix SIGSEGV with  
merge\_tree\_min\_rows\_for\_concurrent\_read/merge\_tree\_min\_bytes\_for\_concurrent\_read=0/UINT64\_MAX.  
#19528 (Azat Khuzhin).
- Buffer overflow (on memory read) was possible if addMonth function was called with specifically crafted arguments. This fixes #19441. This fixes #19413. #19472 (alexey-milovidov).
- Uninitialized memory read was possible in encrypt/decrypt functions if empty string was passed as IV. This closes #19391. #19397 (alexey-milovidov).
- Fix possible buffer overflow in Uber H3 library. See <https://github.com/uber/h3/issues/392>. This closes #19219. #19383 (alexey-milovidov).

- Fix system.parts \_state column (LOGICAL\_ERROR when querying this column, due to incorrect order). [#19346](#) ([Azat Khuzhin](#)).
- Fixed possible wrong result or segfault on aggregation when Materialized View and its target table have different structure. Fixes [#18063](#). [#19322](#) ([tavplubix](#)).
- Fix error Cannot convert column now64() because it is constant but values of constants are different in source and result. Continuation of [#7156](#). [#19316](#) ([Nikolai Kochetov](#)).
- Fix bug when concurrent `ALTER` and `DROP` queries may hang while processing ReplicatedMergeTree table. [#19237](#) ([alesapin](#)).
- Fixed There is no checkpoint error when inserting data through http interface using `Template` or `CustomSeparated` format. Fixes [#19021](#). [#19072](#) ([tavplubix](#)).
- Disable constant folding for subqueries on the analysis stage, when the result cannot be calculated. [#18446](#) ([Azat Khuzhin](#)).
- Mutation might hang waiting for some non-existent part after `MOVE` or `REPLACE PARTITION` or, in rare cases, after `DETACH` or `DROP PARTITION`. It's fixed. [#15537](#) ([tavplubix](#)).

## ClickHouse release v21.1.2.15-stable 2021-01-18

### Backward Incompatible Change

- The setting `input_format_null_as_default` is enabled by default. [#17525](#) ([alexey-milovidov](#)).
- Check settings constraints for profile settings from config. Server will fail to start if `users.xml` contain settings that do not meet corresponding constraints. [#18486](#) ([tavplubix](#)).
- Restrict `ALTER MODIFY SETTING` from changing storage settings that affects data parts (`write_final_mark` and `enable_mixed_granularity_parts`). [#18306](#) ([Amos Bird](#)).
- Set `insert_quorum_parallel` to 1 by default. It is significantly more convenient to use than "sequential" quorum inserts. But if you rely to sequential consistency, you should set the setting back to zero. [#17567](#) ([alexey-milovidov](#)).
- Remove `sumburConsistentHash` function. This closes [#18120](#). [#18656](#) ([alexey-milovidov](#)).
- Removed aggregate functions `timeSeriesGroupSum`, `timeSeriesGroupRateSum` because a friend of mine said they never worked. This fixes [#16869](#). If you have luck using these functions, write a email to [clickhouse-feedback@yandex-team.com](mailto:clickhouse-feedback@yandex-team.com). [#17423](#) ([alexey-milovidov](#)).
- Prohibit `toUnixTimestamp(Date())` (before it just returns `UInt16` representation of Date). [#17376](#) ([Azat Khuzhin](#)).
- Allow using extended integer types (`Int128`, `Int256`, `UInt256`) in `avg` and `avgWeighted` functions. Also allow using different types (integer, decimal, floating point) for value and for weight in `avgWeighted` function. This is a backward-incompatible change: now the `avg` and `avgWeighted` functions always return `Float64` (as documented). Before this change the return type for `Decimal` arguments was also `Decimal`. [#15419](#) ([Mike](#)).
- Expression `toUUID(N)` no longer works. Replace with `toUUID('00000000-0000-0000-0000-000000000000')`. This change is motivated by non-obvious results of `toUUID(N)` where N is non zero.
- SSL Certificates with incorrect "key usage" are rejected. In previous versions they are used to work. See [#19262](#).

- incl references to substitutions file (`/etc/metrika.xml`) were removed from the default config (`<remote_servers>`, `<zookeeper>`, `<macros>`, `<compression>`, `<networks>`). If you were using substitutions file and were relying on those implicit references, you should put them back manually and explicitly by adding corresponding sections with `incl="..."` attributes before the update. See #18740 (alexey-milovidov).

## New Feature

- Implement gRPC protocol in ClickHouse. #15111 (Vitaly Baranov).
- Allow to use multiple zookeeper clusters. #17070 (fastio).
- Implemented `REPLACE TABLE` and `CREATE OR REPLACE TABLE` queries. #18521 (tavplubix).
- Implement `UNION DISTINCT` and treat the plain `UNION` clause as `UNION DISTINCT` by default. Add a setting `union_default_mode` that allows to treat it as `UNION ALL` or require explicit mode specification. #16338 (flynn).
- Added function `accurateCastOrNull`. This closes #10290. Add type conversions in `x IN (subquery)` expressions. This closes #10266. #16724 (Maksim Kita).
- IP Dictionary supports `IPv4` / `IPv6` types directly. #17571 (vdimir).
- IP Dictionary supports key fetching. Resolves #18241. #18480 (vdimir).
- Add `*.zst` compression/decompression support for data import and export. It enables using `*.zst` in `file()` function and `Content-encoding: zstd` in HTTP client. This closes #16791 . #17144 (Abi Palagashvili).
- Added `mannWitneyUTest`, `studentTTest` and `welchTTest` aggregate functions. Refactored `rankCorr` a bit. #16883 (Nikita Mikhaylov).
- Add functions `countMatches`/`countMatchesCaseInsensitive`. #17459 (Azat Khuzhin).
- Implement `countSubstrings()`/`countSubstringsCaseInsensitive()`/`countSubstringsCaseInsensitiveUTF8()` (Count the number of substring occurrences). #17347 (Azat Khuzhin).
- Add information about used databases, tables and columns in `system.query_log`. Add `query_kind` and `normalized_query_hash` fields. #17726 (Amos Bird).
- Add a setting `optimize_on_insert`. When enabled, do the same transformation for `INSERTED` block of data as if `merge` was done on this block (e.g. Replacing, Collapsing, Aggregating...). This setting is enabled by default. This can influence Materialized View and `MaterializeMySQL` behaviour (see detailed description). This closes #10683. #16954 (Kruglov Pavel).
- Kerberos Authentication for HDFS. #16621 (Ilya Golshtain).
- Support `SHOW SETTINGS` statement to show parameters in `system.settings`. `SHOW CHANGED SETTINGS` and `LIKE/ILIKE` clause are also supported. #18056 (Jianmei Zhang).
- Function `position` now supports `POSITION(needle IN haystack)` syntax for SQL compatibility. This closes #18701. ... #18779 (Jianmei Zhang).
- Now we have a new storage setting `max_partitions_to_read` for tables in the MergeTree family. It limits the max number of partitions that can be accessed in one query. A user setting `force_max_partition_limit` is also added to enforce this constraint. #18712 (Amos Bird).
- Add `query_id` column to `system.part_log` for inserted parts. Closes #10097. #18644 (flynn).
- Allow `create table` as `select` with columns specification. Example `CREATE TABLE t1 (x String) ENGINE = Memory AS SELECT 1;`. #18060 (Maksim Kita).

- Added arrayMin, arrayMax, arrayAvg aggregation functions. #18032 (Maksim Kita).
- Implemented ATTACH TABLE name FROM 'path/to/data/' (col1 Type1, ... query. It creates new table with provided structure and attaches table data from provided directory in user\_files. #17903 (tavplubix).
- Add mutation support for StorageMemory. This closes #9117. #15127 (flynn).
- Support syntax EXISTS DATABASE name. #18458 (Du Chuan).
- Support builtin function isIPv4String && isIPv6String like MySQL. #18349 (Du Chuan).
- Add a new setting insert\_distributed\_one\_random\_shard = 1 to allow insertion into multi-sharded distributed table without any distributed key. #18294 (Amos Bird).
- Add settings min\_compress\_block\_size and max\_compress\_block\_size to MergeTreeSettings, which have higher priority than the global settings and take effect when they are set. close 13890. #17867 (flynn).
- Add support for 64bit roaring bitmaps. #17858 (Andy Yang).
- Extended OPTIMIZE ... DEDUPLICATE syntax to allow explicit (or implicit with asterisk/column transformers) list of columns to check for duplicates on. ... #17846 (Vasily Nemkov).
- Added functions toModifiedJulianDay, fromModifiedJulianDay, toModifiedJulianDayOrNull, and fromModifiedJulianDayOrNull. These functions convert between Proleptic Gregorian calendar date and Modified Julian Day number. #17750 (PHO).
- Add ability to use custom TLD list: added functions firstSignificantSubdomainCustom, cutToFirstSignificantSubdomainCustom. #17748 (Azat Khuzhin).
- Add support for PROXYv1 protocol to wrap native TCP interface. Allow quotas to be keyed by proxy-forwarded IP address (applied for PROXYv1 address and for X-Forwarded-For from HTTP interface). This is useful when you provide access to ClickHouse only via trusted proxy (e.g. CloudFlare) but want to account user resources by their original IP addresses. This fixes #17268. #17707 (alexey-milovidov).
- Now clickhouse-client supports opening EDITOR to edit commands. Alt-Shift-E. #17665 (Amos Bird).
- Add function encodeXMLComponent to escape characters to place string into XML text node or attribute. #17659 (nauta).
- Introduce DETACH TABLE/VIEW ... PERMANENTLY syntax, so that after restarting the table does not reappear back automatically on restart (only by explicit request). The table can still be attached back using the short syntax ATTACH TABLE. Implements #5555. Fixes #13850. #17642 (filimonov).
- Add asynchronous metrics on total amount of rows, bytes and parts in MergeTree tables. This fix #11714. #17639 (flynn).
- Add settings limit and offset for out-of-SQL pagination: #16176 They are useful for building APIs. These two settings will affect SELECT query as if it is added like select \* from (your\_original\_select\_query) t limit xxx offset xxxx;. #17633 (hexiaoting).
- Provide a new aggregator combinator : -SimpleState to build SimpleAggregateFunction types via query. It's useful for defining MaterializedView of AggregatingMergeTree engine, and will benefit projections too. #16853 (Amos Bird).
- Added queries-file parameter for clickhouse-client and clickhouse-local. #15930 (Maksim Kita).
- Added query parameter for clickhouse-benchmark. #17832 (Maksim Kita).
- EXPLAIN AST now support queries other then SELECT. #18136 (taiyang-li).

## Experimental Feature

- Added functions for calculation of minHash and simHash of text n-grams and shingles. They are intended for semi-duplicate search. Also functions `bitHammingDistance` and `tupleHammingDistance` are added. #7649 (flynn).
- Add new data type `Map`. See #1841. First version for `Map` only supports `String` type of key and value. #15806 (hexiaoting).
- Implement alternative SQL parser based on ANTLR4 runtime and generated from EBNF grammar. #11298 (ivan).

## Performance Improvement

- New IP Dictionary implementation with lower memory consumption, improved performance for some cases, and fixed bugs. #16804 (vdimir).
- Parallel formatting for data export. #11617 (Nikita Mikhaylov).
- LDAP integration: Added `verification_cooldown` parameter in LDAP server connection configuration to allow caching of successful "bind" attempts for configurable period of time. #15988 (Denis Glazachev).
- Add `--no-system-table` option for `clickhouse-local` to run without system tables. This avoids initialization of `DateLUT` that may take noticeable amount of time (tens of milliseconds) at startup. #18899 (alexey-milovidov).
- Replace `PODArray` with `PODArrayWithStackMemory` in `AggregateFunctionWindowFunnelData` to improve `windowFunnel` function performance. #18817 (flynn).
- Don't send empty blocks to shards on synchronous `INSERT` into Distributed table. This closes #14571. #18775 (alexey-milovidov).
- Optimized read for `StorageMemory`. #18052 (Maksim Kita).
- Using Dragonbox algorithm for float to string conversion instead of ryu. This improves performance of float to string conversion significantly. #17831 (Maksim Kita).
- Speedup `IPv6CIDRToRange` implementation. #17569 (vdimir).
- Add `remerge_sort_lowered_memory_bytes_ratio` setting (If memory usage after remerge does not reduced by this ratio, remerge will be disabled). #17539 (Azat Khuzhin).
- Improve performance of `AggregatingMergeTree` with `SimpleAggregateFunction(String)` in PK. #17109 (Azat Khuzhin).
- Now the `-If` combinator is devirtualized, and `count` is properly vectorized. It is for this PR. #17043 (Amos Bird).
- Fix performance of reading from `Merge` tables over huge number of `MergeTree` tables. Fixes #7748. #16988 (Anton Popov).
- Improved performance of function `repeat`. #16937 (satanson).
- Slightly improved performance of float parsing. #16809 (Maksim Kita).
- Add possibility to skip merged partitions for `OPTIMIZE TABLE ... FINAL`. #15939 (Kruglov Pavel).
- Integrate with `fast_float` from Daniel Lemire to parse floating point numbers. #16787 (Maksim Kita). It is not enabled, because performance its performance is still lower than rough float parser in ClickHouse.
- Fix `max_distributed_connections` (affects `prefer_localhost_replica = 1` and `max_threads != max_distributed_connections`). #17848 (Azat Khuzhin).

- Adaptive choice of single/multi part upload when sending data to S3. Single part upload is controlled by a new setting `max_single_part_upload_size`. #17934 (Pavel Kovalenko).
- Support for async tasks in `PipelineExecutor`. Initial support of async sockets for remote queries. #17868 (Nikolai Kochetov).
- Allow to use `optimize_move_to_prewhere` optimization with compact parts, when sizes of columns are unknown. #17330 (Anton Popov).

## Improvement

- Avoid deadlock when executing `INSERT SELECT` into itself from a table with `TinyLog` or `Log` table engines. This closes #6802. This closes #18691. This closes #16812. This closes #14570. #15260 (alexey-milovidov).
- Support `SHOW CREATE VIEW name` syntax like MySQL. #18095 (Du Chuan).
- All queries of type `Decimal * Float` or vice versa are allowed, including aggregate ones (e.g. `SELECT sum(decimal_field * 1.1)` or `SELECT dec_col * float_col`), the result type is `Float32` or `Float64`. #18145 (Mike).
- Improved minimal Web UI: add history; add sharing support; avoid race condition of different requests; add request in-flight and ready indicators; add favicon; detect Ctrl+Enter if textarea is not in focus. #17293 #17770 (alexey-milovidov).
- `clickhouse-server` didn't send `close` request to ZooKeeper server. #16837 (alesapin).
- Avoid server abnormal termination in case of too low memory limits (`max_memory_usage = 1` / `max_untracked_memory = 1`). #17453 (Azat Khuzhin).
- Fix non-deterministic result of `windowFunnel` function in case of same timestamp for different events. #18884 (Fuwang Hu).
- Docker: Explicitly set uid / gid of clickhouse user & group to the fixed values (101) in `clickhouse-server` Docker images. #19096 (filimonov).
- Asynchronous INSERTs to Distributed tables: Two new settings (by analogy with MergeTree family) has been added: - `fsync_after_insert` - Do fsync for every inserted. Will decreases performance of inserts. - `fsync_directories` - Do fsync for temporary directory (that is used for async INSERT only) after all operations (writes, renames, etc.). #18864 (Azat Khuzhin).
- `SYSTEM KILL` command started to work in Docker. This closes #18847. #18848 (alexey-milovidov).
- Expand macros in the zk path when executing `FETCH PARTITION`. #18839 (fastio).
- Apply `ALTER TABLE <replicated_table> ON CLUSTER MODIFY SETTING ...` to all replicas. Because we don't replicate such alter commands. #18789 (Amos Bird).
- Allow column transformer `EXCEPT` to accept a string as regular expression matcher. This resolves #18685 . #18699 (Amos Bird).
- Fix SimpleAggregateFunction in SummingMergeTree. Now it works like AggregateFunction. In previous versions values were summed together regardless to the aggregate function. This fixes #18564 . #8052. #18637 (Amos Bird). Another fix of using SimpleAggregateFunction in SummingMergeTree. This fixes #18676 . #18677 (Amos Bird).
- Fixed assertion error inside allocator in case when last argument of function bar is NaN. Now simple ClickHouse's exception is being thrown. This fixes #17876. #18520 (Nikita Mikhaylov).
- Fix usability issue: no newline after exception message in some tools. #18444 (alexey-milovidov).

- Add ability to modify primary and partition key column type from `LowCardinality(Type)` to `Type` and vice versa. Also add an ability to modify primary key column type from `EnumX` to `IntX` type. Fixes #5604. #18362 (alesapin).
- Implement `untuple` field access. #18133. #18309 (hexiaoting).
- Allow to parse Array fields from CSV if it is represented as a string containing array that was serialized as nested CSV. Example: `"[\"Hello\", \"world\", \"42\" TV"]"` will parse as `['Hello', 'world', '42' TV']`. Allow to parse array in CSV in a string without enclosing braces. Example: `"Hello, 'world', '42' TV"` will parse as `['Hello', 'world', '42' TV']`. #18271 (alexey-milovidov).
- Make better adaptive granularity calculation for merge tree wide parts. #18223 (alesapin).
- Now `clickhouse install` could work on Mac. The problem was that there is no procfs on this platform. #18201 (Nikita Mikhaylov).
- Better hints for `SHOW ...` query syntax. #18183 (Du Chuan).
- Array aggregation `arrayMin`, `arrayMax`, `arraySum`, `arrayAvg` support for `Int128`, `Int256`, `UInt256`. #18147 (Maksim Kita).
- Add `disk` to Set and Join storage settings. #18112 (Grigory Pervakov).
- Access control: Now table function `merge()` requires current user to have `SELECT` privilege on each table it receives data from. This PR fixes #16964. #18104 #17983 (Vitaly Baranov).
- Temporary tables are visible in the system tables `system.tables` and `system.columns` now only in those session where they have been created. The internal database `_temporary_and_external_tables` is now hidden in those system tables; temporary tables are shown as tables with empty database with the `is_temporary` flag set instead. #18014 (Vitaly Baranov).
- Fix clickhouse-client rendering issue when the size of terminal window changes. #18009 (Amos Bird).
- Decrease log verbosity of the events when the client drops the connection from Warning to Information. #18005 (filimonov).
- Forcibly removing empty or bad metadata files from filesystem for DiskS3. S3 is an experimental feature. #17935 (Pavel Kovalenko).
- Access control: `allow_introspection_functions=0` prohibits usage of introspection functions but doesn't prohibit giving grants for them anymore (the grantee will need to set `allow_introspection_functions=1` for himself to be able to use that grant). Similarly `allow_ddl=0` prohibits usage of DDL commands but doesn't prohibit giving grants for them anymore. #17908 (Vitaly Baranov).
- Usability improvement: hints for column names. #17112. #17857 (fastio).
- Add diagnostic information when two merge tables try to read each other's data. #17854 (徐忻).
- Let the possibility to override timeout value for running script using the ClickHouse docker image. #17818 (Guillaume Tassery).
- Check system log tables' engine definition grammar to prevent some configuration errors. Notes that this grammar check is not semantical, that means such mistakes as non-existent columns / expression functions would be not found out until the table is created. #17739 (Du Chuan).
- Removed exception throwing at RabbitMQ table initialization if there was no connection (it will be reconnecting in the background). #17709 (Kseniia Sumarokova).
- Do not ignore server memory limits during Buffer flush. #17646 (Azat Khuzhin).

- Switch to patched version of RocksDB (from ClickHouse-Extras) to fix use-after-free error. #17643 ([Nikita Mikhaylov](#)).
- Added an offset to exception message for parallel parsing. This fixes #17457. #17641 ([Nikita Mikhaylov](#)).
- Don't throw "Too many parts" error in the middle of INSERT query. #17566 ([alexey-milovidov](#)).
- Allow query parameters in UPDATE statement of ALTER query. Fixes #10976. #17563 ([alexey-milovidov](#)).
- Query obfuscator: avoid usage of some SQL keywords for identifier names. #17526 ([alexey-milovidov](#)).
- Export current max ddl entry executed by DDLWorker via server metric. It's useful to check if DDLWorker hangs somewhere. #17464 ([Amos Bird](#)).
- Export asynchronous metrics of all servers current threads. It's useful to track down issues like [this](#). #17463 ([Amos Bird](#)).
- Include dynamic columns like MATERIALIZED / ALIAS for wildcard query when settings `asterisk_include_materialized_columns` and `asterisk_include_alias_columns` are turned on. #17462 ([Ken Chen](#)).
- Allow specifying TTL to remove old entries from [system log tables](#), using the `<ttl>` attribute in `config.xml`. #17438 ([Du Chuan](#)).
- Now queries coming to the server via MySQL and PostgreSQL protocols have distinctive interface types (which can be seen in the `interface` column of the `tablesystem.query_log`): 4 for MySQL, and 5 for PostgreSQL, instead of formerly used 1 which is now used for the native protocol only. #17437 ([Vitaly Baranov](#)).
- Fix parsing of SETTINGS clause of the `INSERT ... SELECT ... SETTINGS` query. #17414 ([Azat Khuzhin](#)).
- Correctly account memory in RadixSort. #17412 ([Nikita Mikhaylov](#)).
- Add eof check in `receiveHello` in server to prevent getting Attempt to read after eof exception. #17365 ([Kruglov Pavel](#)).
- Avoid possible stack overflow in bigint conversion. Big integers are experimental. #17269 ([flynn](#)).
- Now `set` indices will work with `GLOBAL IN`. This fixes #17232 , #5576 . #17253 ([Amos Bird](#)).
- Add limit for http redirects in request to S3 storage (`s3_max_redirects`). #17220 ([ianton-ru](#)).
- When `-OrNull` combinator combined `-If`, `-Merge`, `-MergeState`, `-State` combinators, we should put `-OrNull` in front. #16935 ([flynn](#)).
- Support HTTP proxy and HTTPS S3 endpoint configuration. #16861 ([Pavel Kovalenko](#)).
- Added proper authentication using environment, `~/.aws` and `AssumeRole` for S3 client. #16856 ([Vladimir Chebotarev](#)).
- Add more OpenTelemetry spans. Add an example of how to export the span data to Zipkin. #16535 ([Alexander Kuzmenkov](#)).
- Cache dictionaries: Completely eliminate callbacks and locks for acquiring them. Keys are not divided into "not found" and "expired", but stored in the same map during query. #14958 ([Nikita Mikhaylov](#)).
- Fix never worked `fsync_part_directory/fsync_after_insert/in_memory_parts_insert_sync` (experimental feature). #18845 ([Azat Khuzhin](#)).
- Allow using Atomic engine for nested database of MaterializeMySQL engine. #14849 ([tavplubix](#)).

## Bug Fix

- Fix the issue when server can stop accepting connections in very rare cases. #17542 (Amos Bird, [alexey-milovidov](#)).
- Fix index analysis of binary functions with constant argument which leads to wrong query results. This fixes #18364. #18373 (Amos Bird).
- Fix possible wrong index analysis when the types of the index comparison are different. This fixes #17122. #17145 (Amos Bird).
- Disable write with AIO during merges because it can lead to extremely rare data corruption of primary key columns during merge. #18481 ([alesapin](#)).
- Restrict merges from wide to compact parts. In case of vertical merge it led to broken result part. #18381 (Anton Popov).
- Fix possible incomplete query result while reading from `MergeTree*` in case of read backoff (message `<Debug> MergeTreeReadPool: Will lower number of threads in logs`). Was introduced in #16423. Fixes #18137. #18216 (Nikolai Kochetov).
- Fix use after free bug in `rocksdb` library. #18862 ([sundyli](#)).
- Fix infinite reading from file in `ORC` format (was introduced in #10580). Fixes #19095. #19134 (Nikolai Kochetov).
- Fix bug in merge tree data writer which can lead to marks with bigger size than fixed granularity size. Fixes #18913. #19123 ([alesapin](#)).
- Fix startup bug when clickhouse was not able to read compression codec from `LowCardinality(Nullable(...))` and throws exception `Attempt to read after EOF`. Fixes #18340. #19101 ([alesapin](#)).
- Restrict `MODIFY TTL` queries for `MergeTree` tables created in old syntax. Previously the query succeeded, but actually it had no effect. #19064 (Anton Popov).
- Make sure `groupUniqArray` returns correct type for argument of `Enum` type. This closes #17875. #19019 ([alexey-milovidov](#)).
- Fix possible error `Expected single dictionary argument for function ignore` with `LowCardinality` argument. Fixes #14275. #19016 (Nikolai Kochetov).
- Fix inserting of `LowCardinality` column to table with `TinyLog` engine. Fixes #18629. #19010 (Nikolai Kochetov).
- Join tries to materialize const columns, but our code wants them in other places. #18982 (Nikita Mikhaylov).
- Disable `optimize_move_functions_out_of_any` because optimization is not always correct. This closes #18051. This closes #18973. #18981 ([alexey-milovidov](#)).
- Fix possible exception `QueryPipeline` stream: different number of columns caused by merging of query plan's `Expression` steps. Fixes #18190. #18980 (Nikolai Kochetov).
- Fixed very rare deadlock at shutdown. #18977 ([tavplubix](#)).
- Fix incorrect behavior when `ALTER TABLE ... DROP PART 'part_name'` query removes all deduplication blocks for the whole partition. Fixes #18874. #18969 ([alesapin](#)).
- Attach partition should reset the mutation. #18804. #18935 ([fastio](#)).

- Fix issue with `bitmapOrCardinality` that may lead to `nullptr` dereference. This closes #18911. #18912 (sundyli).
- Fix possible hang at shutdown in `clickhouse-local`. This fixes #18891. #18893 (alexey-milovidov).
- Queries for external databases (MySQL, ODBC, JDBC) were incorrectly rewritten if there was an expression in form of `x IN table`. This fixes #9756. #18876 (alexey-milovidov).
- Fix \*If combinator with unary function and Nullable types. #18806 (Azat Khuzhin).
- Fix the issue that asynchronous distributed INSERTs can be rejected by the server if the setting `network_compression_method` is globally set to non-default value. This fixes #18741. #18776 (alexey-milovidov).
- Fixed Attempt to read after eof error when trying to `CAST NULL` from `Nullable(String)` to `Nullable(Decimal(P, S))`. Now function `CAST` returns `NULL` when it cannot parse decimal from nullable string. Fixes #7690. #18718 (Winter Zhang).
- Fix minor issue with logging. #18717 (sundyli).
- Fix removing of empty parts in `ReplicatedMergeTree` tables, created with old syntax. Fixes #18582. #18614 (Anton Popov).
- Fix previous bug when date overflow with different values. Strict Date value limit to "2106-02-07", cast date > "2106-02-07" to value 0. #18565 (hexiaoting).
- Add `FixedString` data type support for replication from MySQL. Replication from MySQL is an experimental feature. This patch fixes #18450 Also fixes #6556. #18553 (awesomeleo).
- Fix possible Pipeline stuck error while using `ORDER BY` after subquery with `RIGHT` or `FULL` join. #18550 (Nikolai Kochetov).
- Fix bug which may lead to `ALTER` queries hung after corresponding mutation kill. Found by thread fuzzer. #18518 (alesapin).
- Proper support for 12AM in `parseDateTimeBestEffort` function. This fixes #18402. #18449 (vladimir-golovchenko).
- Fixed value is too short error when executing `toType(...)` functions (`toDate`, `toUInt32`, etc) with argument of type `Nullable(String)`. Now such functions return `NULL` on parsing errors instead of throwing exception. Fixes #7673. #18445 (tavplubix).
- Fix the unexpected behaviour of `SHOW TABLES`. #18431 (fastio).
- Fix -SimpleState combinator generates incompatible arugment type and return type. #18404 (Amos Bird).
- Fix possible race condition in concurrent usage of `Set` or `Join` tables and selects from `system.tables`. #18385 (alexey-milovidov).
- Fix filling table `system.settings_profile_elements`. This PR fixes #18231. #18379 (Vitaly Baranov).
- Fix possible crashes in aggregate functions with combinator `Distinct`, while using two-level aggregation. Fixes #17682. #18365 (Anton Popov).
- Fixed issue when `clickhouse-odbc-bridge` process is unreachable by server on machines with dual IPv4/IPv6 stack; Fixed issue when ODBC dictionary updates are performed using malformed queries and/or cause crashes of the odbc-bridge process; Possibly closes #14489. #18278 (Denis Glazachev).
- Access control: `SELECT count() FROM table` now can be executed if the user has access to at least single column from a table. This PR fixes #10639. #18233 (Vitaly Baranov).

- Access control: `SELECT JOIN` now requires the `SELECT` privilege on each of the joined tables. This PR fixes #17654. #18232 (Vitaly Baranov).
- Fix key comparison between Enum and Int types. This fixes #17989. #18214 (Amos Bird).
- Replication from MySQL (experimental feature). Fixes #18186 Fixes #16372 Fix unique key convert issue in MaterializeMySQL database engine. #18211 (Winter Zhang).
- Fix inconsistency for queries with both `WITH FILL` and `WITH TIES` #17466. #18188 (hexiaoting).
- Fix inserting a row with default value in case of parsing error in the last column. Fixes #17712. #18182 (Jianmei Zhang).
- Fix Unknown setting profile error on attempt to set settings profile. #18167 (tavplubix).
- Fix error when query `MODIFY COLUMN ... REMOVE TTL` doesn't actually remove column TTL. #18130 (alesapin).
- Fixed `std::out_of_range: basic_string` in S3 URL parsing. #18059 (Vladimir Chebotarev).
- Fix comparison of `DateTime64` and `Date`. Fixes #13804 and #11222. ... #18050 (Vasily Nemkov).
- Replication from MySQL (experimental feature): Fixes #15187 Fixes #17912 support convert MySQL prefix index for MaterializeMySQL. #17944 (Winter Zhang).
- When server log rotation was configured using `logger.size` parameter with numeric value larger than  $2^{32}$ , the logs were not rotated properly. This is fixed. #17905 (Alexander Kuzmenkov).
- Trivial query optimization was producing wrong result if query contains `ARRAY JOIN` (so query is actually non trivial). #17887 (sundyli).
- Fix possible segfault in `topK` aggregate function. This closes #17404. #17845 (Maksim Kita).
- WAL (experimental feature): Do not restore parts from WAL if `in_memory_parts_enable_wal` is disabled. #17802 (detaiyang).
- Exception message about max table size to drop was displayed incorrectly. #17764 (alexey-milovidov).
- Fixed possible segfault when there is not enough space when inserting into `Distributed` table. #17737 (tavplubix).
- Fixed problem when ClickHouse fails to resume connection to MySQL servers. #17681 (Alexander Kazakov).
- Windows: Fixed Function not implemented error when executing `RENAME` query in Atomic database with ClickHouse running on Windows Subsystem for Linux. Fixes #17661. #17664 (tavplubix).
- In might be determined incorrectly if cluster is circular- (cross-) replicated or not when executing `ON CLUSTER` query due to race condition when `pool_size > 1`. It's fixed. #17640 (tavplubix).
- Fix empty `system.stack_trace` table when server is running in daemon mode. #17630 (Amos Bird).
- Exception `fmt::v7::format_error` can be logged in background for MergeTree tables. This fixes #17613. #17615 (alexey-milovidov).
- When clickhouse-client is used in interactive mode with multiline queries, single line comment was erroneously extended till the end of query. This fixes #13654. #17565 (alexey-milovidov).
- Fix alter query hang when the corresponding mutation was killed on the different replica. Fixes #16953. #17499 (alesapin).

- Fix issue with memory accounting when mark cache size was underestimated by clickhouse. It may happen when there are a lot of tiny files with marks. #17496 (alesapin).
- Fix ORDER BY with enabled setting `optimize_redundant_functions_in_order_by`. #17471 (Anton Popov).
- Fix duplicates after DISTINCT which were possible because of incorrect optimization. Fixes #17294. #17296 (li chengxiang). #17439 (Nikolai Kochetov).
- Fixed high CPU usage in background tasks of \*MergeTree tables. #17416 (tavplubix).
- Fix possible crash while reading from JOIN table with LowCardinality types. Fixes #17228. #17397 (Nikolai Kochetov).
- Replication from MySQL (experimental feature): Fixes #16835 try fix miss match header with MySQL SHOW statement. #17366 (Winter Zhang).
- Fix nondeterministic functions with predicate optimizer. This fixes #17244. #17273 (Winter Zhang).
- Fix possible Unexpected packet Data received from clienterror for Distributed queries with LIMIT. #17254 (Azat Khuzhin).
- Fix set index invalidation when there are const columns in the subquery. This fixes #17246. #17249 (Amos Bird).
- clickhouse-copier: Fix for non-partitioned tables #15235. #17248 (Qi Chen).
- Fixed possible not-working mutations for parts stored on S3 disk (experimental feature). #17227 (Pavel Kovalenko).
- Bug fix for funciton `fuzzBits`, related issue: #16980. #17051 (hexiaoting).
- Fix `optimize_distributed_group_by_sharding_key` for query with OFFSET only. #16996 (Azat Khuzhin).
- Fix queries from Merge tables over Distributed tables with JOINs. #16993 (Azat Khuzhin).
- Fix order by optimization with monotonic functions. Fixes #16107. #16956 (Anton Popov).
- Fix incorrect comparison of types `DateTime64` with different scales. Fixes #16655 ... #16952 (Vasily Nemkov).
- Fix optimization of group by with enabled setting `optimize_aggregators_of_group_by_keys` and joins. Fixes #12604. #16951 (Anton Popov).
- Minor fix in SHOW ACCESS query. #16866 (tavplubix).
- Fix the behaviour with enabled `optimize_trivial_count_query` setting with partition predicate. #16767 (Azat Khuzhin).
- Return number of affected rows for INSERT queries via MySQL wire protocol. Previously ClickHouse used to always return 0, it's fixed. Fixes #16605. #16715 (Winter Zhang).
- Fix inconsistent behavior caused by `select_sequential_consistency` for optimized trivial count query and system tables. #16309 (Hao Chen).
- Throw error when `REPLACE` column transformer operates on non existing column. #16183 (hexiaoting).
- Throw exception in case of not equi-join ON expression in RIGH|FULL JOIN. #15162 (Artem Zuikov).

## Build/Testing/Packaging Improvement

- Add simple integrity check for ClickHouse binary. It allows to detect corruption due to faulty hardware (bit rot on storage media or bit flips in RAM). #18811 (alexey-milovidov).

- Change `OpenSSL` to `BoringSSL`. It allows to avoid issues with sanitizers. This fixes #12490. This fixes #17502. This fixes #12952. #18129 (alexey-milovidov).
- Simplify `Sys/V` init script. It was not working on Ubuntu 12.04 or older. #17428 (alexey-milovidov).
- Multiple improvements in `./clickhouse install` script. #17421 (alexey-milovidov).
- Now ClickHouse can pretend to be a fake ZooKeeper. Currently, storage implementation is just stored in-memory hash-table, and server partially support ZooKeeper protocol. #16877 (alesapin).
- Fix dead list watches removal for `TestKeeperStorage` (a mock for ZooKeeper). #18065 (alesapin).
- Add `SYSTEM SUSPEND` command for fault injection. It can be used to facilitate failover tests. This closes #15979. #18850 (alexey-milovidov).
- Generate build id when ClickHouse is linked with `lld`. It's appeared that `lld` does not generate it by default on my machine. Build id is used for crash reports and introspection. #18808 (alexey-milovidov).
- Fix shellcheck errors in style check. #18566 (Ilya Yatsishin).
- Update timezones info to 2020e. #18531 (alesapin).
- Fix codespell warnings. Split style checks into separate parts. Update style checks docker image. #18463 (Ilya Yatsishin).
- Automated check for leftovers of conflict markers in docs. #18332 (alexey-milovidov).
- Enable Thread Fuzzer for stateless tests flaky check. #18299 (alesapin).
- Do not use non thread-safe function `strerror`. #18204 (alexey-milovidov).
- Update `anchore/scan-action@main` workflow action (was moved from `master` to `main`). #18192 (Stig Bakken).
- Now `clickhouse-test` does `DROP/CREATE` databases with a timeout. #18098 (alesapin).
- Enable experimental support for Pytest framework for stateless tests. #17902 (Ivan).
- Now we use the fresh docker daemon version in integration tests. #17671 (alesapin).
- Send info about official build, memory, cpu and free disk space to Sentry if it is enabled. Sentry is opt-in feature to help ClickHouse developers. This closes #17279. #17543 (alexey-milovidov).
- There was an uninitialized variable in the code of `clickhouse-copier`. #17363 (Nikita Mikhaylov).
- Fix one MSan report from #17309. #17344 (Nikita Mikhaylov).
- Fix for the issue with IPv6 in Arrow Flight library. See the comments for details. #16664 (Zhanna).
- Add a library that replaces some `libc` functions to traps that will terminate the process. #16366 (alexey-milovidov).
- Provide diagnostics in server logs in case of stack overflow, send error message to `clickhouse-client`. This closes #14840. #16346 (alexey-milovidov).
- Now we can run almost all stateless functional tests in parallel. #15236 (alesapin).
- Fix corruption in `librdkafka` snappy decompression (was a problem only for gcc10 builds, but official builds uses clang already, so at least recent official releases are not affected). #18053 (Azat Khuzhin).
- If server was terminated by OOM killer, print message in log. #13516 (alexey-milovidov).
- PODArray: Avoid call to `memcpy` with `(nullptr, 0)` arguments (Fix UBSan report). This fixes #18525. #18526 (alexey-milovidov).

- Minor improvement for path concatenation of zookeeper paths inside DDLWorker. [#17767](#) ([Bharat Nallan](#)).
- Allow to reload symbols from debug file. This PR also fixes a build-id issue. [#17637](#) ([Amos Bird](#)).

## Changelog for 2020

---

### ClickHouse release 20.12

#### ClickHouse release v20.12.5.14-stable, 2020-12-28

##### Bug Fix

- Disable write with AIO during merges because it can lead to extremely rare data corruption of primary key columns during merge. [#18481](#) ([alesapin](#)).
- Fixed value is too short error when executing `toType(...)` functions (`toDate`, `toUInt32`, etc) with argument of type `Nullable(String)`. Now such functions return `NULL` on parsing errors instead of throwing exception. Fixes [#7673](#). [#18445](#) ([tavplubix](#)).
- Restrict merges from wide to compact parts. In case of vertical merge it led to broken result part. [#18381](#) ([Anton Popov](#)).
- Fix filling table `system.settings_profile_elements`. This PR fixes [#18231](#). [#18379](#) ([Vitaly Baranov](#)).
- Fix possible crashes in aggregate functions with combinator `Distinct`, while using two-level aggregation. Fixes [#17682](#). [#18365](#) ([Anton Popov](#)).
- Fix error when query `MODIFY COLUMN ... REMOVE TTL` does not actually remove column TTL. [#18130](#) ([alesapin](#)).

##### Build/Testing/Packaging Improvement

- Update timezones info to 2020e. [#18531](#) ([alesapin](#)).

#### ClickHouse release v20.12.4.5-stable, 2020-12-24

##### Bug Fix

- Fixed issue when `clickhouse-odbc-bridge` process is unreachable by server on machines with dual IPv4/IPv6 stack; - Fixed issue when ODBC dictionary updates are performed using malformed queries and/or cause crashes; Possibly closes [#14489](#). [#18278](#) ([Denis Glazachev](#)).
- Fixed key comparison between `Enum` and `Int` types. This fixes [#17989](#). [#18214](#) ([Amos Bird](#)).
- Fixed unique key convert crash in `MaterializeMySQL` database engine. This fixes [#18186](#) and fixes [#16372](#). [#18211](#) ([Winter Zhang](#)).
- Fixed `std::out_of_range`: `basic_string` in S3 URL parsing. [#18059](#) ([Vladimir Chebotarev](#)).
- Fixed the issue when some tables not synchronized to ClickHouse from MySQL caused by the fact that conversion MySQL prefix index wasn't supported for `MaterializeMySQL`. This fixes [#15187](#) and fixes [#17912](#) [#17944](#) ([Winter Zhang](#)).
- Fixed the issue when query optimization was producing wrong result if query contains `ARRAY JOIN`. [#17887](#) ([sundylj](#)).
- Fixed possible segfault in `topK` aggregate function. This closes [#17404](#). [#17845](#) ([Maksim Kita](#)).
- Fixed empty `system.stack_trace` table when server is running in daemon mode. [#17630](#) ([Amos Bird](#)).

# ClickHouse release v20.12.3.3-stable, 2020-12-13

## Backward Incompatible Change

- Enable `use_compact_format_in_distributed_parts_names` by default (see the documentation for the reference). [#16728 \(Azat Khuzhin\)](#).
- Accept user settings related to file formats (e.g. `format_csv_delimiter`) in the `SETTINGS` clause when creating a table that uses `File` engine, and use these settings in all `INSERTs` and `SELECTs`. The file format settings changed in the current user session, or in the `SETTINGS` clause of a DML query itself, no longer affect the query. [#16591 \(Alexander Kuzmenkov\)](#).

## New Feature

- add `*.xz` compression/decompression support. It enables using `*.xz` in `file()` function. This closes [#8828](#). [#16578 \(Abi Palagashvili\)](#).
- Introduce the query `ALTER TABLE ... DROP|DETACH PART 'part_name'`. [#15511 \(nvartolomei\)](#).
- Added new `ALTER UPDATE/DELETE IN PARTITION` syntax. [#13403 \(Vladimir Chebotarev\)](#).
- Allow formatting named tuples as JSON objects when using JSON input/output formats, controlled by the `output_format_json_named_tuples_as_objects` setting, disabled by default. [#17175 \(Alexander Kuzmenkov\)](#).
- Add a possibility to input enum value as its id in TSV and CSV formats by default. [#16834 \(Kruglov Pavel\)](#).
- Add COLLATE support for Nullable, LowCardinality, Array and Tuple, where nested type is String. Also refactor the code associated with collations in `ColumnString.cpp`. [#16273 \(Kruglov Pavel\)](#).
- New `tcpPort` function returns TCP port listened by this server. [#17134 \(Ivan\)](#).
- Add new math functions: `acosh`, `asinh`, `atan2`, `atanh`, `cosh`, `hypot`, `log1p`, `sinh`. [#16636 \(Konstantin Malanchev\)](#).
- Possibility to distribute the merges between different replicas. Introduces the `execute_merges_on_single_replica_time_threshold` mergetree setting. [#16424 \(filimonov\)](#).
- Add setting `aggregate_functions_null_for_empty` for SQL standard compatibility. This option will rewrite all aggregate functions in a query, adding `-OrNull` suffix to them. Implements [10273](#). [#16123 \(flynn\)](#).
- Updated `DateTime`, `DateTime64` parsing to accept string Date literal format. [#16040 \(Maksim Kita\)](#).
- Make it possible to change the path to history file in `clickhouse-client` using the `--history_file` parameter. [#15960 \(Maksim Kita\)](#).

## Bug Fix

- Fix the issue when server can stop accepting connections in very rare cases. [#17542 \(Amos Bird\)](#).
- Fixed `Function not implemented` error when executing `RENAME` query in `Atomic` database with ClickHouse running on Windows Subsystem for Linux. Fixes [#17661](#). [#17664 \(tavplubix\)](#).
- Do not restore parts from WAL if `in_memory_parts_enable_wal` is disabled. [#17802 \(detailyang\)](#).
- fix incorrect initialization of `max_compress_block_size` of `MergeTreeWriterSettings` with `min_compress_block_size`. [#17833 \(flynn\)](#).
- Exception message about max table size to drop was displayed incorrectly. [#17764 \(alexey-milovidov\)](#).
- Fixed possible segfault when there is not enough space when inserting into Distributed table. [#17737 \(tavplubix\)](#).

- Fixed problem when ClickHouse fails to resume connection to MySQL servers. #17681 (Alexander Kazakov).
- In might be determined incorrectly if cluster is circular- (cross-) replicated or not when executing `ON CLUSTER` query due to race condition when `pool_size > 1`. It's fixed. #17640 (tavplubix).
- Exception `fmt::v7::format_error` can be logged in background for MergeTree tables. This fixes #17613. #17615 (alexey-milovidov).
- When clickhouse-client is used in interactive mode with multiline queries, single line comment was erroneously extended till the end of query. This fixes #13654. #17565 (alexey-milovidov).
- Fix alter query hang when the corresponding mutation was killed on the different replica. Fixes #16953. #17499 (alesapin).
- Fix issue when mark cache size was underestimated by clickhouse. It may happen when there are a lot of tiny files with marks. #17496 (alesapin).
- Fix `ORDER BY` with enabled setting `optimize_redundant_functions_in_order_by`. #17471 (Anton Popov).
- Fix duplicates after `DISTINCT` which were possible because of incorrect optimization. Fixes #17294. #17296 (li chengxiang). #17439 (Nikolai Kochetov).
- Fix crash while reading from `JOIN` table with `LowCardinality` types. Fixes #17228. #17397 (Nikolai Kochetov).
- fix `toInt256(inf)` stack overflow. Int256 is an experimental feature. Closed #17235. #17257 (flynn).
- Fix possible `Unexpected packet Data received from clienterror` logged for Distributed queries with `LIMIT`. #17254 (Azat Khuzhin).
- Fix set index invalidation when there are const columns in the subquery. This fixes #17246. #17249 (Amos Bird).
- Fix possible wrong index analysis when the types of the index comparison are different. This fixes #17122. #17145 (Amos Bird).
- Fix ColumnConst comparison which leads to crash. This fixed #17088 . #17135 (Amos Bird).
- Multiple fixed for MaterializeMySQL (experimental feature). Fixes #16923 Fixes #15883 Fix MaterializeMySQL SYNC failure when the modify MySQL binlog\_checksum. #17091 (Winter Zhang).
- Fix bug when `ON CLUSTER` queries may hang forever for non-leader ReplicatedMergeTreeTables. #17089 (alesapin).
- Fixed crash on `CREATE TABLE ... AS some_table` query when `some_table` was created `AS table_function()` Fixes #16944. #17072 (tavplubix).
- Bug unfinished implementation for funciton fuzzBits, related issue: #16980. #17051 (hexiaoting).
- Fix LLVM's libunwind in the case when CFA register is RAX. This is the `bug` in LLVM's libunwind. We already have workarounds for this bug. #17046 (alexey-milovidov).
- Avoid unnecessary network errors for remote queries which may be cancelled while execution, like queries with `LIMIT`. #17006 (Azat Khuzhin).
- Fix `optimize_distributed_group_by_sharding_key` setting (that is disabled by default) for query with `OFFSET` only. #16996 (Azat Khuzhin).
- Fix for Merge tables over Distributed tables with `JOIN`. #16993 (Azat Khuzhin).

- Fixed wrong result in big integers (128, 256 bit) when casting from double. Big integers support is experimental. [#16986 \(Mike\)](#).
- Fix possible server crash after `ALTER TABLE ... MODIFY COLUMN ... NewType` when `SELECT` have `WHERE` expression on altering column and alter does not finished yet. [#16968 \(Amos Bird\)](#).
- Blame info was not calculated correctly in `clickhouse-git-import`. [#16959 \(alexey-milovidov\)](#).
- Fix order by optimization with monotonous functions. Fixes [#16107](#). [#16956 \(Anton Popov\)](#).
- Fix optimization of group by with enabled setting `optimize_aggregators_of_group_by_keys` and joins. Fixes [#12604](#). [#16951 \(Anton Popov\)](#).
- Fix possible error `Illegal type of argument` for queries with `ORDER BY`. Fixes [#16580](#). [#16928 \(Nikolai Kochetov\)](#).
- Fix strange code in `InterpreterShowAccessQuery`. [#16866 \(tavplubix\)](#).
- Prevent clickhouse server crashes when using the function `timeSeriesGroupSum`. The function is removed from newer ClickHouse releases. [#16865 \(filimonov\)](#).
- Fix rare silent crashes when query profiler is on and ClickHouse is installed on OS with glibc version that has (supposedly) broken asynchronous unwind tables for some functions. This fixes [#15301](#). This fixes [#13098](#). [#16846 \(alexey-milovidov\)](#).
- Fix crash when using `any` without any arguments. This is for [#16803](#). cc @azat. [#16826 \(Amos Bird\)](#).
- If no memory can be allocated while writing table metadata on disk, broken metadata file can be written. [#16772 \(alexey-milovidov\)](#).
- Fix trivial query optimization with partition predicate. [#16767 \(Azat Khuzhin\)](#).
- Fix `IN` operator over several columns and tuples with enabled `transform_null_in` setting. Fixes [#15310](#). [#16722 \(Anton Popov\)](#).
- Return number of affected rows for `INSERT` queries via MySQL protocol. Previously ClickHouse used to always return 0, it's fixed. Fixes [#16605](#). [#16715 \(Winter Zhang\)](#).
- Fix remote query failure when using 'if' suffix aggregate function. Fixes [#16574](#) Fixes [#16231](#) [#16610 \(Winter Zhang\)](#).
- Fix inconsistent behavior caused by `select_sequential_consistency` for optimized trivial count query and `system.tables`. [#16309 \(Hao Chen\)](#).

## Improvement

- Remove empty parts after they were pruned by TTL, mutation, or collapsing merge algorithm. [#16895 \(Anton Popov\)](#).
- Enable compact format of directories for asynchronous sends in Distributed tables: `use_compact_format_in_distributed_parts_names` is set to 1 by default. [#16788 \(Azat Khuzhin\)](#).
- Abort multipart upload if no data was written to S3. [#16840 \(Pavel Kovalenko\)](#).
- Reresolve the IP of the `format_avro_schema_registry_url` in case of errors. [#16985 \(filimonov\)](#).
- Mask password in `data_path` in the `system.distribution_queue`. [#16727 \(Azat Khuzhin\)](#).
- Throw error when use column transformer replaces non existing column. [#16183 \(hexiaoting\)](#).

- Turn off parallel parsing when there is no enough memory for all threads to work simultaneously. Also there could be exceptions like "Memory limit exceeded" when somebody will try to insert extremely huge rows (> min\_chunk\_bytes\_for\_parallel\_parsing), because each piece to parse has to be independent set of strings (one or more). [#16721](#) ([Nikita Mikhaylov](#)).
- Install script should always create subdirs in config folders. This is only relevant for Docker build with custom config. [#16936](#) ([filimonov](#)).
- Correct grammar in error message in JSONEachRow, JSONCompactEachRow, and RegexpRow input formats. [#17205](#) ([nico piderman](#)).
- Set default host and port parameters for SOURCE(CLICKHOUSE(...)) to current instance and set default user value to 'default'. [#16997](#) ([vdimir](#)).
- Throw an informative error message when doing ATTACH/DETACH TABLE <DICTIONARY>. Before this PR, detach table <dict> works but leads to an ill-formed in-memory metadata. [#16885](#) ([Amos Bird](#)).
- Add cutToFirstSignificantSubdomainWithWWW(). [#16845](#) ([Azat Khuzhin](#)).
- Server refused to startup with exception message if wrong config is given (metric\_log.collect\_interval\_milliseconds is missing). [#16815](#) ([Ivan](#)).
- Better exception message when configuration for distributed DDL is absent. This fixes [#5075](#). [#16769](#) ([Nikita Mikhaylov](#)).
- Usability improvement: better suggestions in syntax error message when CODEC expression is misplaced in CREATE TABLE query. This fixes [#12493](#). [#16768](#) ([alexey-milovidov](#)).
- Remove empty directories for async INSERT at start of Distributed engine. [#16729](#) ([Azat Khuzhin](#)).
- Workaround for use S3 with nginx server as proxy. Nginx currently does not accept urls with empty path like http://domain.com?delete, but vanilla aws-sdk-cpp produces this kind of urls. This commit uses patched aws-sdk-cpp version, which makes urls with "/" as path in this cases, like http://domain.com/?delete. [#16709](#) ([ianton-ru](#)).
- Allow reinterpretAs\* functions to work for integers and floats of the same size. Implements [16640](#). [#16657](#) ([flynn](#)).
- Now, <auxiliary\_zookeepers> configuration can be changed in config.xml and reloaded without server startup. [#16627](#) ([Amos Bird](#)).
- Support SNI in https connections to remote resources. This will allow to connect to Cloudflare servers that require SNI. This fixes [#10055](#). [#16252](#) ([alexey-milovidov](#)).
- Make it possible to connect to clickhouse-server secure endpoint which requires SNI. This is possible when clickhouse-server is hosted behind TLS proxy. [#16938](#) ([filimonov](#)).
- Fix possible stack overflow if a loop of materialized views is created. This closes [#15732](#). [#16048](#) ([alexey-milovidov](#)).
- Simplify the implementation of background tasks processing for the MergeTree table engines family. There should be no visible changes for user. [#15983](#) ([alesapin](#)).
- Improvement for MaterializeMySQL (experimental feature). Throw exception about right sync privileges when MySQL sync user has error privileges. [#15977](#) ([TCeason](#)).
- Made indexOf() use BloomFilter. [#14977](#) ([achimbab](#)).

## Performance Improvement

- Use Floyd-Rivest algorithm, it is the best for the ClickHouse use case of partial sorting. Benchmarks are in <https://github.com/danlark1/miniselect> and here. #16825 (Danila Kutenin).
- Now ReplicatedMergeTree tree engines family uses a separate thread pool for replicated fetches. Size of the pool limited by setting `background_fetches_pool_size` which can be tuned with a server restart. The default value of the setting is 3 and it means that the maximum amount of parallel fetches is equal to 3 (and it allows to utilize 10G network). Fixes #520. #16390 (alesapin).
- Fixed uncontrolled growth of the state of quantileTDigest. #16680 (hrissan).
- Add `VIEW` subquery description to `EXPLAIN`. Limit push down optimisation for `VIEW`. Add local replicas of `Distributed` to query plan. #14936 (Nikolai Kochetov).
- Fix `optimize_read_in_order`/`optimize_aggregation_in_order` with `max_threads > 0` and expression in `ORDER BY`. #16637 (Azat Khuzhin).
- Fix performance of reading from `Merge` tables over huge number of `MergeTree` tables. Fixes #7748. #16988 (Anton Popov).
- Now we can safely prune partitions with exact match. Useful case: Suppose table is partitioned by `intHash64(x) % 100` and the query has condition on `intHash64(x) % 100` verbatim, not on `x`. #16253 (Amos Bird).

## Experimental Feature

- Add `EmbeddedRocksDB` table engine (can be used for dictionaries). #15073 (sundyli).

## Build/Testing/Packaging Improvement

- Improvements in test coverage building images. #17233 (alesapin).
- Update embedded timezone data to version 2020d (also update cctz to the latest master). #17204 (filimonov).
- Fix UBSan report in Poco. This closes #12719. #16765 (alexey-milovidov).
- Do not instrument 3rd-party libraries with UBSan. #16764 (alexey-milovidov).
- Fix UBSan report in cache dictionaries. This closes #12641. #16763 (alexey-milovidov).
- Fix UBSan report when trying to convert infinite floating point number to integer. This closes #14190. #16677 (alexey-milovidov).

# ClickHouse release 20.11

## ClickHouse release v20.11.7.16-stable, 2021-03-02

### Improvement

- Explicitly set uid / gid of clickhouse user & group to the fixed values (101) in clickhouse-server images. #19096 (filimonov).

### Bug Fix

- BloomFilter index crash fix. Fixes #19757. #19884 (Maksim Kita).
- Deadlock was possible if `system.text_log` is enabled. This fixes #19874. #19875 (alexey-milovidov).
- In previous versions, unusual arguments for function `arrayEnumerateUniq` may cause crash or infinite loop. This closes #19787. #19788 (alexey-milovidov).

- Fixed stack overflow when using accurate comparison of arithmetic type with string type. #19773 ([tavplubix](#)).
- Fix a segmentation fault in `bitmapAndnot` function. Fixes #19668. #19713 ([Maksim Kita](#)).
- Some functions with big integers may cause segfault. Big integers is experimental feature. This closes #19667. #19672 ([alexey-milovidov](#)).
- Fix wrong result of function `neighbor` for `LowCardinality` argument. Fixes #10333. #19617 ([Nikolai Kochetov](#)).
- Fix use-after-free of the `CompressedWriteBuffer` in `Connection` after disconnect. #19599 ([Azat Khuzhin](#)).
- `DROP/DETACH TABLE` table ON CLUSTER cluster `SYNC` query might hang, it's fixed. Fixes #19568. #19572 ([tavplubix](#)).
- Query `CREATE DICTIONARY` id expression fix. #19571 ([Maksim Kita](#)).
- Fix SIGSEGV with  
`merge_tree_min_rows_for_concurrent_read/merge_tree_min_bytes_for_concurrent_read=0/UINT64_MAX.`  
#19528 ([Azat Khuzhin](#)).
- Buffer overflow (on memory read) was possible if `addMonth` function was called with specifically crafted arguments. This fixes #19441. This fixes #19413. #19472 ([alexey-milovidov](#)).
- Mark distributed batch as broken in case of empty data block in one of files. #19449 ([Azat Khuzhin](#)).
- Fix possible buffer overflow in Uber H3 library. See <https://github.com/uber/h3/issues/392>. This closes #19219. #19383 ([alexey-milovidov](#)).
- Fix system.parts \_state column (LOGICAL\_ERROR when querying this column, due to incorrect order). #19346 ([Azat Khuzhin](#)).
- Fix error `Cannot convert column now64()` because it is constant but values of constants are different in source and result. Continuation of #7156. #19316 ([Nikolai Kochetov](#)).
- Fix bug when concurrent `ALTER` and `DROP` queries may hang while processing `ReplicatedMergeTree` table. #19237 ([alesapin](#)).
- Fix infinite reading from file in `ORC` format (was introduced in #10580). Fixes #19095. #19134 ([Nikolai Kochetov](#)).
- Fix startup bug when clickhouse was not able to read compression codec from `LowCardinality(Nullable(...))` and throws exception `Attempt to read after EOF`. Fixes #18340. #19101 ([alesapin](#)).
- Fixed `There is no checkpoint` error when inserting data through http interface using `Template` or `CustomSeparated` format. Fixes #19021. #19072 ([tavplubix](#)).
- Restrict `MODIFY TTL` queries for `MergeTree` tables created in old syntax. Previously the query succeeded, but actually it had no effect. #19064 ([Anton Popov](#)).
- Make sure `groupUniqArray` returns correct type for argument of `Enum` type. This closes #17875. #19019 ([alexey-milovidov](#)).
- Fix possible error `Expected single dictionary argument for function ignore` if use function `ignore` with `LowCardinality` argument. Fixes #14275. #19016 ([Nikolai Kochetov](#)).
- Fix inserting of `LowCardinality` column to table with `TinyLog` engine. Fixes #18629. #19010 ([Nikolai Kochetov](#)).

- Disable `optimize_move_functions_out_of_any` because optimization is not always correct. This closes [#18051](#). This closes [#18973](#). [#18981](#) ([alexey-milovidov](#)).
- Fixed very rare deadlock at shutdown. [#18977](#) ([tavplubix](#)).
- Fix bug when mutation with some escaped text (like `ALTER ... UPDATE e = CAST('foo', 'Enum8(\\"foo\\" = 1')`) serialized incorrectly. Fixes [#18878](#). [#18944](#) ([alesapin](#)).
- Attach partition should reset the mutation. [#18804](#). [#18935](#) ([fastio](#)).
- Fix possible hang at shutdown in `clickhouse-local`. This fixes [#18891](#). [#18893](#) ([alexey-milovidov](#)).
- Fix \*If combinator with unary function and Nullable types. [#18806](#) ([Azat Khuzhin](#)).
- Asynchronous distributed INSERTs can be rejected by the server if the setting `network_compression_method` is globally set to non-default value. This fixes [#18741](#). [#18776](#) ([alexey-milovidov](#)).
- Fixed Attempt to read after eof error when trying to `CAST NULL` from `Nullable(String)` to `Nullable(Decimal(P, S))`. Now function `CAST` returns `NULL` when it cannot parse decimal from nullable string. Fixes [#7690](#). [#18718](#) ([Winter Zhang](#)).
- Fix Logger with unmatched arg size. [#18717](#) ([sundyli](#)).
- Add `FixedString` Data type support. I'll get this exception "Code: 50, e.displayText() = DB::Exception: Unsupported type `FixedString(1)`" when replicating data from MySQL to ClickHouse. This patch fixes bug [#18450](#) Also fixes [#6556](#). [#18553](#) ([awesomeléo](#)).
- Fix possible `Pipeline` stuck error while using `ORDER BY` after subquery with `RIGHT` or `FULL` join. [#18550](#) ([Nikolai Kochetov](#)).
- Fix bug which may lead to `ALTER` queries hung after corresponding mutation kill. Found by thread fuzzer. [#18518](#) ([alesapin](#)).
- Disable write with AIO during merges because it can lead to extremely rare data corruption of primary key columns during merge. [#18481](#) ([alesapin](#)).
- Disable constant folding for subqueries on the analysis stage, when the result cannot be calculated. [#18446](#) ([Azat Khuzhin](#)).
- Fixed value is too short error when executing `toType(...)` functions (`toDate`, `toUInt32`, etc) with argument of type `Nullable(String)`. Now such functions return `NULL` on parsing errors instead of throwing exception. Fixes [#7673](#). [#18445](#) ([tavplubix](#)).
- Restrict merges from wide to compact parts. In case of vertical merge it led to broken result part. [#18381](#) ([Anton Popov](#)).
- Fix filling table `system.settings_profile_elements`. This PR fixes [#18231](#). [#18379](#) ([Vitaly Baranov](#)).
- Fix index analysis of binary functions with constant argument which leads to wrong query results. This fixes [#18364](#). [#18373](#) ([Amos Bird](#)).
- Fix possible crashes in aggregate functions with combinator `Distinct`, while using two-level aggregation. Fixes [#17682](#). [#18365](#) ([Anton Popov](#)).
- `SELECT count()` FROM table now can be executed if only one any column can be selected from the table. This PR fixes [#10639](#). [#18233](#) ([Vitaly Baranov](#)).
- `SELECT JOIN` now requires the `SELECT` privilege on each of the joined tables. This PR fixes [#17654](#). [#18232](#) ([Vitaly Baranov](#)).

- Fix possible incomplete query result while reading from `MergeTree*` in case of read backoff (message `<Debug> MergeTreeReadPool: Will lower number of threads` in logs). Was introduced in #16423. Fixes #18137. #18216 (Nikolai Kochetov).
- Fix error when query `MODIFY COLUMN ... REMOVE TTL` does not actually remove column TTL. #18130 (alesapin).
- Fix indeterministic functions with predicate optimizer. This fixes #17244. #17273 (Winter Zhang).
- Mutation might hang waiting for some non-existent part after `MOVE` or `REPLACE PARTITION` or, in rare cases, after `DETACH` or `DROP PARTITION`. It's fixed. #15537 (tavplubix).

## Build/Testing/Packaging Improvement

- Update timezones info to 2020e. #18531 (alesapin).

## ClickHouse release v20.11.6.6-stable, 2020-12-24

### Bug Fix

- Fixed issue when `clickhouse-odbc-bridge` process is unreachable by server on machines with dual IPv4/IPv6 stack and fixed issue when ODBC dictionary updates are performed using malformed queries and/or cause crashes. This possibly closes #14489. #18278 (Denis Glazachev).
- Fixed key comparison between `Enum` and `Int` types. This fixes #17989. #18214 (Amos Bird).
- Fixed unique key convert crash in `MaterializeMySQL` database engine. This fixes #18186 and fixes #16372 #18211 (Winter Zhang).
- Fixed `std::out_of_range: basic_string` in S3 URL parsing. #18059 (Vladimir Chebotarev).
- Fixed the issue when some tables not synchronized to ClickHouse from MySQL caused by the fact that conversion MySQL prefix index wasn't supported for `MaterializeMySQL`. This fixes #15187 and fixes #17912 #17944 (Winter Zhang).
- Fixed the issue when query optimization was producing wrong result if query contains `ARRAY JOIN`. #17887 (sundyli).
- Fix possible segfault in `topK` aggregate function. This closes #17404. #17845 (Maksim Kita).
- Do not restore parts from WAL if `in_memory_parts_enable_wal` is disabled. #17802 (detailyang).
- Fixed problem when ClickHouse fails to resume connection to MySQL servers. #17681 (Alexander Kazakov).
- Fixed inconsistent behaviour of `optimize_trivial_count_query` with partition predicate. #17644 (Azat Khuzhin).
- Fixed empty `system.stack_trace` table when server is running in daemon mode. #17630 (Amos Bird).
- Fixed the behaviour when `xxception fmt::v7::format_error` can be logged in background for `MergeTree` tables. This fixes #17613. #17615 (alexey-milovidov).
- Fixed the behaviour when `clickhouse-client` is used in interactive mode with multiline queries and single line comment was erroneously extended till the end of query. This fixes #13654. #17565 (alexey-milovidov).
- Fixed the issue when server can stop accepting connections in very rare cases. #17542 (alexey-milovidov).
- Fixed alter query hang when the corresponding mutation was killed on the different replica. This fixes #16953. #17499 (alesapin).

- Fixed bug when mark cache size was underestimated by clickhouse. It may happen when there are a lot of tiny files with marks. [#17496](#) ([alesapin](#)).
- Fixed ORDER BY with enabled setting `optimize_redundant_functions_in_order_by`. [#17471](#) ([Anton Popov](#)).
- Fixed duplicates after `DISTINCT` which were possible because of incorrect optimization. This fixes [#17294](#). [#17296](#) ([li chengxiang](#)). [#17439](#) ([Nikolai Kochetov](#)).
- Fixed crash while reading from JOIN table with LowCardinality types. This fixes [#17228](#). [#17397](#) ([Nikolai Kochetov](#)).
- Fixed set index invalidation when there are const columns in the subquery. This fixes [#17246](#) . [#17249](#) ([Amos Bird](#)).
- Fixed possible wrong index analysis when the types of the index comparison are different. This fixes [#17122](#). [#17145](#) ([Amos Bird](#)).
- Fixed `ColumnConst` comparison which leads to crash. This fixes [#17088](#) . [#17135](#) ([Amos Bird](#)).
- Fixed bug when `ON CLUSTER` queries may hang forever for non-leader `ReplicatedMergeTreeTables`. [#17089](#) ([alesapin](#)).
- Fixed fuzzer-found bug in funciton `fuzzBits`. This fixes [#16980](#). [#17051](#) ([hexiaoting](#)).
- Avoid unnecessary network errors for remote queries which may be cancelled while execution, like queries with `LIMIT`. [#17006](#) ([Azat Khuzhin](#)).
- Fixed wrong result in big integers (128, 256 bit) when casting from double. [#16986](#) ([Mike](#)).
- Reresolve the IP of the `format_avro_schema_registry_url` in case of errors. [#16985](#) ([filimonov](#)).
- Fixed possible server crash after `ALTER TABLE ... MODIFY COLUMN ... NewType`when `SELECT` have `WHERE` expression on altering column and alter does not finished yet. [#16968](#) ([Amos Bird](#)).
- Blame info was not calculated correctly in `clickhouse-git-import`. [#16959](#) ([alexey-milovidov](#)).
- Fixed order by optimization with monotonous functions. Fixes [#16107](#). [#16956](#) ([Anton Popov](#)).
- Fixed optimization of group by with enabled setting `optimize_aggregators_of_group_by_keys` and joins. This fixes [#12604](#). [#16951](#) ([Anton Popov](#)).
- Install script should always create subdirs in config folders. This is only relevant for Docker build with custom config. [#16936](#) ([filimonov](#)).
- Fixed possible error `Illegal type of argument for queries with ORDER BY`. This fixes [#16580](#). [#16928](#) ([Nikolai Kochetov](#)).
- Abort multipart upload if no data was written to `WriteBufferFromS3`. [#16840](#) ([Pavel Kovalenko](#)).
- Fixed crash when using `any` without any arguments. This fixes [#16803](#). [#16826](#) ([Amos Bird](#)).
- Fixed the behaviour when ClickHouse used to always return 0 insted of a number of affected rows for `INSERT` queries via MySQL protocol. This fixes [#16605](#). [#16715](#) ([Winter Zhang](#)).
- Fixed uncontrolled growth of TDigest. [#16680](#) ([hrissan](#)).
- Fixed remote query failure when using suffix `if` in Aggregate function. This fixes [#16574](#) fixes [#16231](#) [#16610](#) ([Winter Zhang](#)).
- Fixed inconsistent behavior caused by `select_sequential_consistency` for optimized trivial count query and system.tables. [#16309](#) ([Hao Chen](#)).

- Throw error when use ColumnTransformer replace non exist column. #16183 (hexiaoting).

## ClickHouse release v20.11.3.3-stable, 2020-11-13

### Bug Fix

- Fix rare silent crashes when query profiler is on and ClickHouse is installed on OS with glibc version that has (supposedly) broken asynchronous unwind tables for some functions. This fixes #15301. This fixes #13098. #16846 (alexey-milovidov).

## ClickHouse release v20.11.2.1, 2020-11-11

### Backward Incompatible Change

- If some `profile` was specified in `distributed_ddl` config section, then this profile could overwrite settings of `default` profile on server startup. It's fixed, now settings of distributed DDL queries should not affect global server settings. #16635 (tavplubix).
- Restrict to use of non-comparable data types (like `AggregateFunction`) in keys (Sorting key, Primary key, Partition key, and so on). #16601 (alesapin).
- Remove `ANALYZE` and `AST` queries, and make the setting `enable_debug_queries` obsolete since now it is the part of full featured `EXPLAIN` query. #16536 (Ivan).
- Aggregate functions `boundingRatio`, `rankCorr`, `retention`, `timeSeriesGroupSum`, `timeSeriesGroupRateSum`, `windowFunnel` were erroneously made case-insensitive. Now their names are made case sensitive as designed. Only functions that are specified in SQL standard or made for compatibility with other DBMS or functions similar to those should be case-insensitive. #16407 (alexey-milovidov).
- Make `rankCorr` function return nan on insufficient data #16124. #16135 (hexiaoting).
- When upgrading from versions older than 20.5, if rolling update is performed and cluster contains both versions 20.5 or greater and less than 20.5, if ClickHouse nodes with old versions are restarted and old version has been started up in presence of newer versions, it may lead to `Part ... intersects previous part` errors. To prevent this error, first install newer clickhouse-server packages on all cluster nodes and then do restarts (so, when clickhouse-server is restarted, it will start up with the new version).

### New Feature

- Added support of LDAP as a user directory for locally non-existent users. #12736 (Denis Glazachev).
- Add `system.replicated_fetches` table which shows currently running background fetches. #16428 (alesapin).
- Added setting `date_time_output_format`. #15845 (Maksim Kita).
- Added minimal web UI to ClickHouse. #16158 (alexey-milovidov).
- Allows to read/write Single protobuf message at once (w/o length-delimiters). #15199 (filimonov).
- Added initial OpenTelemetry support. ClickHouse now accepts OpenTelemetry traceparent headers over Native and HTTP protocols, and passes them downstream in some cases. The trace spans for executed queries are saved into the `system.opentelemetry_span_log` table. #14195 (Alexander Kuzmenkov).
- Allow specify primary key in column list of `CREATE TABLE` query. This is needed for compatibility with other SQL dialects. #15823 (Maksim Kita).
- Implement `OFFSET offset_row_count {ROW | ROWS} FETCH {FIRST | NEXT} fetch_row_count {ROW | ROWS} {ONLY | WITH TIES}` in `SELECT` query with `ORDER BY`. This is the SQL-standard way to specify `LIMIT`. #15855 (hexiaoting).

- `errorCodeToName` function - return variable name of the error (useful for analyzing `query_log` and similar). `system.errors` table - shows how many times errors has been happened (respects `system_events_show_zero_values`). [#16438 \(Azat Khuzhin\)](#).
- Added function `untuple` which is a special function which can introduce new columns to the `SELECT` list by expanding a named tuple. [#16242 \(Nikolai Kochetov, Amos Bird\)](#).
- Now we can provide identifiers via query parameters. And these parameters can be used as table objects or columns. [#16594 \(Amos Bird\)](#).
- Added big integers (`UInt256`, `Int128`, `Int256`) and UUID data types support for MergeTree BloomFilter index. Big integers is an experimental feature. [#16642 \(Maksim Kita\)](#).
- Add `farmFingerprint64` function (non-cryptographic string hashing). [#16570 \(Jacob Hayes\)](#).
- Add `log_queries_min_query_duration_ms`, only queries slower than the value of this setting will go to `query_log/query_thread_log` (i.e. something like `slow_query_log` in mysql). [#16529 \(Azat Khuzhin\)](#).
- Ability to create a docker image on the top of `Alpine`. Uses precompiled binary and glibc components from ubuntu 20.04. [#16479 \(filimonov\)](#).
- Added `toUUIDOrNull`, `toUUIDOrZero` cast functions. [#16337 \(Maksim Kita\)](#).
- Add `max_concurrent_queries_for_all_users` setting, see [#6636](#) for use cases. [#16154 \(nvartolomei\)](#).
- Add a new option `print_query_id` to `clickhouse-client`. It helps generate arbitrary strings with the current query id generated by the client. Also print query id in `clickhouse-client` by default. [#15809 \(Amos Bird\)](#).
- Add `tid` and `logTrace` functions. This closes [#9434](#). [#15803 \(flynn\)](#).
- Add function `formatReadableTimeDelta` that format time delta to human readable string ... [#15497 \(Filipe Caixeta\)](#).
- Added `disable_merges` option for volumes in multi-disk configuration. [#13956 \(Vladimir Chebotarev\)](#).

## Experimental Feature

- New functions `encrypt`, `aes_encrypt_mysql`, `decrypt`, `aes_decrypt_mysql`. These functions are working slowly, so we consider it as an experimental feature. [#11844 \(Vasily Nemkov\)](#).

## Bug Fix

- Mask password in `data_path` in the `system.distribution_queue`. [#16727 \(Azat Khuzhin\)](#).
- Fix `IN` operator over several columns and tuples with enabled `transform_null_in` setting. Fixes [#15310](#). [#16722 \(Anton Popov\)](#).
- The setting `max_parallel_replicas` worked incorrectly if the queried table has no sampling. This fixes [#5733](#). [#16675 \(alexey-milovidov\)](#).
- Fix `optimize_read_in_order/optimize_aggregation_in_order` with `max_threads > 0` and expression in `ORDER BY`. [#16637 \(Azat Khuzhin\)](#).
- Calculation of `DEFAULT` expressions was involving possible name collisions (that was very unlikely to encounter). This fixes [#9359](#). [#16612 \(alexey-milovidov\)](#).
- Fix `query_thread_log.query_duration_ms` unit. [#16563 \(Azat Khuzhin\)](#).
- Fix a bug when using MySQL Master -> MySQL Slave -> ClickHouse MaterializeMySQL Engine. `MaterializeMySQL` is an experimental feature. [#16504 \(TCeason\)](#).

- Specifically crafted argument of `round` function with `Decimal` was leading to integer division by zero. This fixes #13338. #16451 (alexey-milovidov).
- Fix DROP TABLE for Distributed (racy with INSERT). #16409 (Azat Khuzhin).
- Fix processing of very large entries in replication queue. Very large entries may appear in ALTER queries if table structure is extremely large (near 1 MB). This fixes #16307. #16332 (alexey-milovidov).
- Fixed the inconsistent behaviour when a part of return data could be dropped because the set for its filtration wasn't created. #16308 (Nikita Mikhaylov).
- Fix dictGet in sharding\_key (and similar places, i.e. when the function context is stored permanently). #16205 (Azat Khuzhin).
- Fix the exception thrown in `clickhouse-local` when trying to execute `OPTIMIZE` command. Fixes #16076. #16192 (filimonov).
- Fixes #15780 regression, e.g. `indexOf([1, 2, 3], toLowCardinality(1))` now is prohibited but it should not be. #16038 (Mike).
- Fix bug with MySQL database. When MySQL server used as database engine is down some queries raise Exception, because they try to get tables from disabled server, while it's unnecessary. For example, query `SELECT ... FROM system.parts` should work only with MergeTree tables and don't touch MySQL database at all. #16032 (Kruglov Pavel).
- Now exception will be thrown when `ALTER MODIFY COLUMN ... DEFAULT ...` has incompatible default with column type. Fixes #15854. #15858 (alesapin).
- Fixed IPv4CIDRToRange/IPv6CIDRToRange functions to accept const IP-column values. #15856 (vladimir-golovchenko).

## Improvement

- Treat `INTERVAL '1 hour'` as equivalent to `INTERVAL 1 HOUR`, to be compatible with Postgres and similar. This fixes #15637. #15978 (flynn).
- Enable parsing enum values by their numeric ids for CSV, TSV and JSON input formats. #15685 (vivarum).
- Better read task scheduling for JBOD architecture and `MergeTree` storage. New setting `read_backoff_min_concurrency` which serves as the lower limit to the number of reading threads. #16423 (Amos Bird).
- Add missing support for `LowCardinality` in `Avro` format. #16521 (Mike).
- Workaround for use `S3` with `nginx` server as proxy. Nginx currently does not accept urls with empty path like `http://domain.com?delete`, but vanilla `aws-sdk-cpp` produces this kind of urls. This commit uses patched `aws-sdk-cpp` version, which makes urls with "/" as path in this cases, like `http://domain.com/?delete`. #16814 (ianton-ru).
- Better diagnostics on parse errors in input data. Provide row number on `Cannot read all data` errors. #16644 (alexey-milovidov).
- Make the behaviour of `minMap` and `maxMap` more desirable. It will not skip zero values in the result. Fixes #16087. #16631 (Ildus Kurbangaliev).
- Better update of ZooKeeper configuration in runtime. #16630 (sundyl).
- Apply SETTINGS clause as early as possible. It allows to modify more settings in the query. This closes #3178. #16619 (alexey-milovidov).

- Now `event_time_microseconds` field stores in Decimal64, not UInt64. #16617 (Nikita Mikhaylov).
- Now parameterized functions can be used in `APPLY` column transformer. #16589 (Amos Bird).
- Improve scheduling of background task which removes data of dropped tables in Atomic databases. Atomic databases do not create broken symlink to table data directory if table actually has no data directory. #16584 (tavplubix).
- Subqueries in `WITH` section (CTE) can reference previous subqueries in `WITH` section by their name. #16575 (Amos Bird).
- Add `current_database` into `system.query_thread_log`. #16558 (Azat Khuzhin).
- Allow to fetch parts that are already committed or outdated in the current instance into the detached directory. It's useful when migrating tables from another cluster and having N to 1 shards mapping. It's also consistent with the current `fetchPartition` implementation. #16538 (Amos Bird).
- Multiple improvements for RabbitMQ: Fixed bug for #16263. Also minimized event loop lifetime. Added more efficient queues setup. #16426 (Ksenia Sumarokova).
- Fix debug assertion in `quantileDeterministic` function. In previous version it may also transfer up to two times more data over the network. Although no bug existed. This fixes #15683. #16410 (alexey-milovidov).
- Add `TablesToDeleteQueueSize` metric. It's equal to number of dropped tables, that are waiting for background data removal. #16364 (tavplubix).
- Better diagnostics when client has dropped connection. In previous versions, `Attempt to read after EOF` and `Broken pipe` exceptions were logged in server. In new version, it's information message `Client has dropped the connection, cancel the query..` #16329 (alexey-milovidov).
- Add `total_rows/total_bytes` (from `system.tables`) support for Set/Join table engines. #16306 (Azat Khuzhin).
- Now it's possible to specify `PRIMARY KEY` without `ORDER BY` for MergeTree table engines family. Closes #15591. #16284 (alesapin).
- If there is no `tmp` folder in the system (chroot, misconfiguration etc) `clickhouse-local` will create temporary subfolder in the current directory. #16280 (filimonov).
- Add support for nested data types (like named tuple) as sub-types. Fixes #15587. #16262 (Ivan).
- Support for `database_atomic_wait_for_drop_and_detach_synchronously/NO DELAY/SYNC` for `DROP DATABASE`. #16127 (Azat Khuzhin).
- Add `allow_nondeterministic_optimize_skip_unused_shards` (to allow non deterministic like `rand()` or `dictGet()` in sharding key). #16105 (Azat Khuzhin).
- Fix `memory_profiler_step/max.untracked_memory` for queries via HTTP (test included). Fix the issue that adjusting this value globally in xml config does not help either, since those settings are not applied anyway, only default (4MB) value is `used`. Fix `query_id` for the most root ThreadStatus of the http query (by initializing `QueryScope` after reading `query_id`). #16101 (Azat Khuzhin).
- Now it's allowed to execute `ALTER ... ON CLUSTER` queries regardless of the `<internal_replication>` setting in cluster config. #16075 (alesapin).
- Fix rare issue when `clickhouse-client` may abort on exit due to loading of suggestions. This fixes #16035. #16047 (alexey-milovidov).
- Add support of cache layout for Redis dictionaries with complex key. #15985 (Anton Popov).

- Fix query hang (endless loop) in case of misconfiguration (`connections_with_failover_max_tries` set to 0). [#15876 \(Azat Khuzhin\)](#).
- Change level of some log messages from information to debug, so information messages will not appear for every query. This closes [#5293](#). [#15816 \(alexey-milovidov\)](#).
- Remove `MemoryTrackingInBackground*` metrics to avoid potentially misleading results. This fixes [#15684](#). [#15813 \(alexey-milovidov\)](#).
- Add reconnects to `zookeeper-dump-tree` tool. [#15711 \(alexey-milovidov\)](#).
- Allow explicitly specify columns list in `CREATE TABLE table AS table_function(...)` query. Fixes [#9249](#) Fixes [#14214](#). [#14295 \(tavplubix\)](#).

## Performance Improvement

- Do not merge parts across partitions in `SELECT FINAL`. [#15938 \(Kruglov Pavel\)](#).
- Improve performance of `-OrNull` and `-OrDefault` aggregate functions. [#16661 \(alexey-milovidov\)](#).
- Improve performance of `quantileMerge`. In previous versions it was obviously slow. This closes [#1463](#). [#16643 \(alexey-milovidov\)](#).
- Improve performance of logical functions a little. [#16347 \(alexey-milovidov\)](#).
- Improved performance of merges assignment in MergeTree table engines. Shouldn't be visible for the user. [#16191 \(alesapin\)](#).
- Speedup hashed/sparse\_hashed dictionary loading by preallocating the hash table. [#15454 \(Azat Khuzhin\)](#).
- Now trivial count optimization becomes slightly non-trivial. Predicates that contain exact partition expr can be optimized too. This also fixes [#11092](#) which returns wrong count when `max_parallel_replicas > 1`. [#15074 \(Amos Bird\)](#).

## Build/Testing/Packaging Improvement

- Add flaky check for stateless tests. It will detect potentially flaky functional tests in advance, before they are merged. [#16238 \(alesapin\)](#).
- Use proper version for `croaring` instead of amalgamation. [#16285 \(sundyli\)](#).
- Improve generation of build files for `ya.make` build system (Arcadia). [#16700 \(alexey-milovidov\)](#).
- Add MySQL BinLog file check tool for `MaterializeMySQL` database engine. `MaterializeMySQL` is an experimental feature. [#16223 \(Winter Zhang\)](#).
- Check for executable bit on non-executable files. People often accidentally commit executable files from Windows. [#15843 \(alexey-milovidov\)](#).
- Check for `#pragma once` in headers. [#15818 \(alexey-milovidov\)](#).
- Fix illegal code style `&vector[idx]` in `libhdfs3`. This fixes libcxx debug build. See also <https://github.com/ClickHouse-Extras/libhdfs3/pull/8> . [#15815 \(Amos Bird\)](#).
- Fix build of one miscellaneous example tool on Mac OS. Note that we don't build examples on Mac OS in our CI (we build only ClickHouse binary), so there is zero chance it will not break again. This fixes [#15804](#). [#15808 \(alexey-milovidov\)](#).
- Simplify Sys/V init script. [#14135 \(alexey-milovidov\)](#).

- Added `boost::program_options` to `db_generator` in order to increase its usability. This closes #15940. #15973 (Nikita Mikhaylov).

## ClickHouse release 20.10

### ClickHouse release v20.10.7.4-stable, 2020-12-24

#### Bug Fix

- Fixed issue when `clickhouse-odbc-bridge` process is unreachable by server on machines with dual IPv4/IPv6 stack and fixed issue when ODBC dictionary updates are performed using malformed queries and/or cause crashes. This possibly closes #14489. #18278 (Denis Glazachev).
- Fix key comparison between `Enum` and `Int` types. This fixes #17989. #18214 (Amos Bird).
- Fixed unique key convert crash in `MaterializeMySQL` database engine. This fixes #18186 and fixes #16372 #18211 (Winter Zhang).
- Fixed `std::out_of_range: basic_string` in S3 URL parsing. #18059 (Vladimir Chebotarev).
- Fixed the issue when some tables not synchronized to ClickHouse from MySQL caused by the fact that conversion MySQL prefix index wasn't supported for `MaterializeMySQL`. This fixes #15187 and fixes #17912 #17944 (Winter Zhang).
- Fix possible segfault in `topK` aggregate function. This closes #17404. #17845 (Maksim Kita).
- Do not restore parts from `WAL` if `in_memory_parts_enable_wal` is disabled. #17802 (detailyang).
- Fixed problem when ClickHouse fails to resume connection to MySQL servers. #17681 (Alexander Kazakov).
- Fixed empty `system.stack_trace` table when server is running in daemon mode. #17630 (Amos Bird).
- Fixed the behaviour when `clickhouse-client` is used in interactive mode with multiline queries and single line comment was erroneously extended till the end of query. This fixes #13654. #17565 (alexey-milovidov).
- Fixed the issue when server can stop accepting connections in very rare cases. #17542 (alexey-milovidov).
- Fixed `ALTER` query hang when the corresponding mutation was killed on the different replica. This fixes #16953. #17499 (alesapin).
- Fixed bug when mark cache size was underestimated by clickhouse. It may happen when there are a lot of tiny files with marks. #17496 (alesapin).
- Fixed ORDER BY with enabled setting `optimize_redundant_functions_in_order_by`. #17471 (Anton Popov).
- Fixed duplicates after `DISTINCT` which were possible because of incorrect optimization. Fixes #17294. #17296 (li chengxiang). #17439 (Nikolai Kochetov).
- Fixed crash while reading from `JOIN` table with `LowCardinality` types. This fixes #17228. #17397 (Nikolai Kochetov).
- Fixed set index invalidation when there are const columns in the subquery. This fixes #17246 . #17249 (Amos Bird).
- Fixed `ColumnConst` comparison which leads to crash. This fixed #17088 . #17135 (Amos Bird).
- Fixed bug when ON CLUSTER queries may hang forever for non-leader ReplicatedMergeTreeTables. #17089 (alesapin).

- Fixed fuzzer-found bug in function `fuzzBits`. This fixes #16980. #17051 (hexiaoting).
- Avoid unnecessary network errors for remote queries which may be cancelled while execution, like queries with `LIMIT`. #17006 (Azat Khuzhin).
- Fixed wrong result in big integers (128, 256 bit) when casting from double. #16986 (Mike).
- Reresolve the IP of the `format_avro_schema_registry_url` in case of errors. #16985 (filimonov).
- Fixed possible server crash after `ALTER TABLE ... MODIFY COLUMN ... NewType` when `SELECT` have `WHERE` expression on altering column and alter does not finished yet. #16968 (Amos Bird).
- Blame info was not calculated correctly in `clickhouse-git-import`. #16959 (alexey-milovidov).
- Fixed order by optimization with monotonous functions. This fixes #16107. #16956 (Anton Popov).
- Fixrf optimization of group by with enabled setting `optimize_aggregators_of_group_by_keys` and joins. This fixes #12604. #16951 (Anton Popov).
- Install script should always create subdirs in config folders. This is only relevant for Docker build with custom config. #16936 (filimonov).
- Fixrf possible error `Illegal type of argument` for queries with `ORDER BY`. This fixes #16580. #16928 (Nikolai Kochetov).
- Abort multipart upload if no data was written to `WriteBufferFromS3`. #16840 (Pavel Kovalenko).
- Fixed crash when using `any` without any arguments. This fixes #16803. #16826 (Amos Bird).
- Fixed the behaviour when ClickHouse used to always return 0 instead of a number of affected rows for `INSERT` queries via MySQL protocol. This fixes #16605. #16715 (Winter Zhang).
- Fixed uncontrolled growth of `TDigest`. #16680 (hrissan).
- Fixed remote query failure when using suffix `if` in Aggregate function. This fixes #16574 fixes #16231 #16610 (Winter Zhang).

## ClickHouse release v20.10.4.1-stable, 2020-11-13

### Bug Fix

- Fix rare silent crashes when query profiler is on and ClickHouse is installed on OS with glibc version that has (supposedly) broken asynchronous unwind tables for some functions. This fixes #15301. This fixes #13098. #16846 (alexey-milovidov).
- Fix `IN` operator over several columns and tuples with enabled `transform_null_in` setting. Fixes #15310. #16722 (Anton Popov).
- This will fix `optimize_read_in_order/optimize_aggregation_in_order` with `max_threads>0` and expression in `ORDER BY`. #16637 (Azat Khuzhin).
- Now when parsing AVRO from input the `LowCardinality` is removed from type. Fixes #16188. #16521 (Mike).
- Fix rapid growth of metadata when using MySQL Master -> MySQL Slave -> ClickHouse MaterializeMySQL Engine, and `slave_parallel_worker` enabled on MySQL Slave, by properly shrinking GTID sets. This fixes #15951. #16504 (TCeason).
- Fix `DROP TABLE` for Distributed (racy with `INSERT`). #16409 (Azat Khuzhin).
- Fix processing of very large entries in replication queue. Very large entries may appear in `ALTER` queries if table structure is extremely large (near 1 MB). This fixes #16307. #16332 (alexey-milovidov).

- Fix bug with MySQL database. When MySQL server used as database engine is down some queries raise Exception, because they try to get tables from disabled server, while it's unnecessary. For example, query `SELECT ... FROM system.parts` should work only with MergeTree tables and don't touch MySQL database at all. [#16032 \(Kruglov Pavel\)](#).

## Improvement

- Workaround for use S3 with nginx server as proxy. Nginx currently does not accept urls with empty path like <http://domain.com?delete>, but vanilla aws-sdk-cpp produces this kind of urls. This commit uses patched aws-sdk-cpp version, which makes urls with "/" as path in this cases, like <http://domain.com/?delete>. [#16813 \(ianton-ru\)](#).

## ClickHouse release v20.10.3.30, 2020-10-28

### Backward Incompatible Change

- Make `multiple_joins_rewriter_version` obsolete. Remove first version of joins rewriter. [#15472 \(Artem Zuikov\)](#).
- Change default value of `format_regexpEscapingRule` setting (it's related to `Regexp` format) to `Raw` (it means - read whole subpattern as a value) to make the behaviour more like to what users expect. [#15426 \(alexey-milovidov\)](#).
- Add support for nested multiline comments `/* comment /* comment */ */` in SQL. This conforms to the SQL standard. [#14655 \(alexey-milovidov\)](#).
- Added MergeTree settings (`max_replicated_merges_with_ttl_in_queue` and `max_number_of_merges_with_ttl_in_pool`) to control the number of merges with TTL in the background pool and replicated queue. This change breaks compatibility with older versions only if you use delete TTL. Otherwise, replication will stay compatible. You can avoid incompatibility issues if you update all shard replicas at once or execute `SYSTEM STOP TTL MERGES` until you finish the update of all replicas. If you'll get an incompatible entry in the replication queue, first of all, execute `SYSTEM STOP TTL MERGES` and after `ALTER TABLE ... DETACH PARTITION ...` the partition where incompatible TTL merge was assigned. Attach it back on a single replica. [#14490 \(alesapin\)](#).
- When upgrading from versions older than 20.5, if rolling update is performed and cluster contains both versions 20.5 or greater and less than 20.5, if ClickHouse nodes with old versions are restarted and old version has been started up in presence of newer versions, it may lead to `Part ... intersects previous part` errors. To prevent this error, first install newer clickhouse-server packages on all cluster nodes and then do restarts (so, when clickhouse-server is restarted, it will start up with the new version).

### New Feature

- Background data recompression. Add the ability to specify `TTL ... RECOMPRESS codec_name` for MergeTree table engines family. [#14494 \(alesapin\)](#).
- Add parallel quorum inserts. This closes [#15601](#). [#15601 \(Latysheva Alexandra\)](#).
- Settings for additional enforcement of data durability. Useful for non-replicated setups. [#11948 \(Anton Popov\)](#).
- When duplicate block is written to replica where it does not exist locally (has not been fetched from replicas), don't ignore it and write locally to achieve the same effect as if it was successfully replicated. [#11684 \(alexey-milovidov\)](#).
- Now we support `WITH <identifier> AS (subquery) ...` to introduce named subqueries in the query context. This closes [#2416](#). This closes [#4967](#). [#14771 \(Amos Bird\)](#).

- Introduce `enable_global_with_statement` setting which propagates the first select's `WITH` statements to other select queries at the same level, and makes aliases in `WITH` statements visible to subqueries. #15451 (Amos Bird).
- Secure inter-cluster query execution (with `initial_user` as current query user). #13156 (Azat Khuzhin). #15551 (Azat Khuzhin).
- Add the ability to remove column properties and table TTLs. Introduced queries `ALTER TABLE MODIFY COLUMN col_name REMOVE what_to_remove` and `ALTER TABLE REMOVE TTL`. Both operations are lightweight and executed at the metadata level. #14742 (alesapin).
- Added format `RawBLOB`. It is intended for input or output a single value without any escaping and delimiters. This closes #15349. #15364 (alexey-milovidov).
- Add the `reinterpretAsUUID` function that allows to convert a big-endian byte string to UUID. #15480 (Alexander Kuzmenkov).
- Implement `force_data_skipping_indices` setting. #15642 (Azat Khuzhin).
- Add a setting `output_format_pretty_row_numbers` to numerate the result in Pretty formats. This closes #15350. #15443 (flynn).
- Added query obfuscation tool. It allows to share more queries for better testing. This closes #15268. #15321 (alexey-milovidov).
- Add table function `null('structure')`. #14797 (vxider).
- Added `formatReadableQuantity` function. It is useful for reading big numbers by human. #14725 (Artem Hnilov).
- Add format `LineAsString` that accepts a sequence of lines separated by newlines, every line is parsed as a whole as a single String field. #14703 (Nikita Mikhaylov), #13846 (hexiaoting).
- Add `JSONStrings` format which output data in arrays of strings. #14333 (hc).
- Add support for "Raw" column format for `Regexp` format. It allows to simply extract subpatterns as a whole without any escaping rules. #15363 (alexey-milovidov).
- Allow configurable `NULL` representation for `TSV` output format. It is controlled by the setting `output_format_tsv_null_representation` which is `\N` by default. This closes #9375. Note that the setting only controls output format and `\N` is the only supported `NULL` representation for `TSV` input format. #14586 (Kruglov Pavel).
- Support Decimal data type for `MaterializeMySQL`. `MaterializeMySQL` is an experimental feature. #14535 (Winter Zhang).
- Add new feature: `SHOW DATABASES LIKE 'xxx'`. #14521 (hexiaoting).
- Added a script to import (arbitrary) git repository to ClickHouse as a sample dataset. #14471 (alexey-milovidov).
- Now insert statements can have asterisk (or variants) with column transformers in the column list. #14453 (Amos Bird).
- New query complexity limit settings `max_rows_to_read_leaf`, `max_bytes_to_read_leaf` for distributed queries to limit max rows/bytes read on the leaf nodes. Limit is applied for local reads only, *excluding* the final merge stage on the root node. #14221 (Roman Khavronenko).

- Allow user to specify settings for ReplicatedMergeTree\* storage in `<replicated_merge_tree>` section of config file. It works similarly to `<merge_tree>` section. For ReplicatedMergeTree\* storages settings from `<merge_tree>` and `<replicated_merge_tree>` are applied together, but settings from `<replicated_merge_tree>` has higher priority. Added system.replicated\_merge\_tree\_settings table. #13573 (Amos Bird).
- Add `mapPopulateSeries` function. #13166 (Ildus Kurbangaliev).
- Supporting MySQL types: `decimal` (as ClickHouse `Decimal`) and `datetime` with sub-second precision (as `DateTime64`). #11512 (Vasily Nemkov).
- Introduce `event_time_microseconds` field to `system.text_log`, `system.trace_log`, `system.query_log` and `system.query_thread_log` tables. #14760 (Bharat Nallan).
- Add `event_time_microseconds` to `system.asynchronous_metric_log` & `system.metric_log` tables. #14514 (Bharat Nallan).
- Add `query_start_time_microseconds` field to `system.query_log` & `system.query_thread_log` tables. #14252 (Bharat Nallan).

## Bug Fix

- Fix the case when memory can be overallocated regardless to the limit. This closes #14560. #16206 (alexey-milovidov).
- Fix `executable` dictionary source hang. In previous versions, when using some formats (e.g. `JSONEachRow`) data was not feed to a child process before it outputs at least something. This closes #1697. This closes #2455. #14525 (alexey-milovidov).
- Fix double free in case of exception in function `dictGet`. It could have happened if dictionary was loaded with error. #16429 (Nikolai Kochetov).
- Fix group by with totals/rollup/cube modifiers and min/max functions over group by keys. Fixes #16393. #16397 (Anton Popov).
- Fix async Distributed INSERT with `prefer_localhost_replica=0` and `internal_replication`. #16358 (Azat Khuzhin).
- Fix a very wrong code in `TwoLevelStringHashTable` implementation, which might lead to memory leak. #16264 (Amos Bird).
- Fix segfault in some cases of wrong aggregation in lambdas. #16082 (Anton Popov).
- Fix `ALTER MODIFY ... ORDER BY` query hang for `ReplicatedVersionedCollapsingMergeTree`. This fixes #15980. #16011 (alesapin).
- `MaterializeMySQL` (experimental feature): Fix collate name & charset name parser and support `length = 0` for string type. #16008 (Winter Zhang).
- Allow to use `direct` layout for dictionaries with complex keys. #16007 (Anton Popov).
- Prevent replica hang for 5-10 mins when replication error happens after a period of inactivity. #15987 (filimonov).
- Fix rare segfaults when inserting into or selecting from `MaterializedView` and concurrently dropping target table (for Atomic database engine). #15984 (tavplubix).
- Fix ambiguity in parsing of settings profiles: `CREATE USER ... SETTINGS profile readonly` is now considered as using a profile named `readonly`, not a setting named `profile` with the `readonly` constraint. This fixes #15628. #15982 (Vitaly Baranov).
- `MaterializeMySQL` (experimental feature): Fix crash on create database failure. #15954 (Winter Zhang).

- Fixed `DROP TABLE IF EXISTS` failure with `Table ... does not exist` error when table is concurrently renamed (for Atomic database engine). Fixed rare deadlock when concurrently executing some DDL queries with multiple tables (like `DROP DATABASE` and `RENAME TABLE`) - Fixed `DROP/DETACH DATABASE` failure with `Table ... does not exist` when concurrently executing `DROP/DETACH TABLE`. [#15934 \(tavplubix\)](#).
- Fix incorrect empty result for query from Distributed table if query has `WHERE`, `PREWHERE` and `GLOBAL IN`. Fixes [#15792](#). [#15933 \(Nikolai Kochetov\)](#).
- Fixes [#12513](#): difference expressions with same alias when query is reanalyzed. [#15886 \(Winter Zhang\)](#).
- Fix possible very rare deadlocks in RBAC implementation. [#15875 \(Vitaly Baranov\)](#).
- Fix exception `Block structure mismatch` in `SELECT ... ORDER BY DESC` queries which were executed after `ALTER MODIFY COLUMN` query. Fixes [#15800](#). [#15852 \(alesapin\)](#).
- MaterializeMySQL (experimental feature): Fix `select count()` inaccuracy. [#15767 \(tavplubix\)](#).
- Fix some cases of queries, in which only virtual columns are selected. Previously `Not found column _nothing in block` exception may be thrown. Fixes [#12298](#). [#15756 \(Anton Popov\)](#).
- Fix drop of materialized view with inner table in Atomic database (hangs all subsequent `DROP TABLE` due to hang of the worker thread, due to recursive `DROP TABLE` for inner table of MV). [#15743 \(Azat Khuzhin\)](#).
- Possibility to move part to another disk/volume if the first attempt was failed. [#15723 \(Pavel Kovalenko\)](#).
- Fix error `Cannot find column` which may happen at insertion into `MATERIALIZED VIEW` in case if query for `MV` contains `ARRAY JOIN`. [#15717 \(Nikolai Kochetov\)](#).
- Fixed too low default value of `max_replicated_logs_to_keep` setting, which might cause replicas to become lost too often. Improve lost replica recovery process by choosing the most up-to-date replica to clone. Also do not remove old parts from lost replica, detach them instead. [#15701 \(tavplubix\)](#).
- Fix rare race condition in dictionaries and tables from MySQL. [#15686 \(alesapin\)](#).
- Fix (benign) race condition in AMQP-CPP. [#15667 \(alesapin\)](#).
- Fix error `Cannot add simple transform to empty Pipe` which happened while reading from `Buffer` table which has different structure than destination table. It was possible if destination table returned empty result for query. Fixes [#15529](#). [#15662 \(Nikolai Kochetov\)](#).
- Proper error handling during insert into MergeTree with S3. MergeTree over S3 is an experimental feature. [#15657 \(Pavel Kovalenko\)](#).
- Fixed bug with S3 table function: region from URL was not applied to S3 client configuration. [#15646 \(Vladimir Chebotarev\)](#).
- Fix the order of destruction for resources in `ReadFromStorage` step of query plan. It might cause crashes in rare cases. Possibly connected with [#15610](#). [#15645 \(Nikolai Kochetov\)](#).
- Subtract `ReadonlyReplica` metric when detach readonly tables. [#15592 \(sundyli\)](#).
- Fixed `Element ... is not a constant expression` error when using `JSON*` function result in `VALUES`, `LIMIT` or right side of `IN` operator. [#15589 \(tavplubix\)](#).
- Query will finish faster in case of exception. Cancel execution on remote replicas if exception happens. [#15578 \(Azat Khuzhin\)](#).
- Prevent the possibility of error message `Could not calculate available disk space (statvfs), errno: 4, strerror: Interrupted system call`. This fixes [#15541](#). [#15557 \(alexey-milovidov\)](#).

- Fix Database <db> does not exist. in queries with IN and Distributed table when there's no database on initiator. [#15538 \(Artem Zuikov\)](#).
- Mutation might hang waiting for some non-existent part after MOVE or REPLACE PARTITION or, in rare cases, after DETACH or DROP PARTITION. It's fixed. [#15537 \(tavplubix\)](#).
- Fix bug when ILIKE operator stops being case insensitive if LIKE with the same pattern was executed. [#15536 \(alesapin\)](#).
- Fix Missing columns errors when selecting columns which absent in data, but depend on other columns which also absent in data. Fixes [#15530](#). [#15532 \(alesapin\)](#).
- Throw an error when a single parameter is passed to ReplicatedMergeTree instead of ignoring it. [#15516 \(nvartolomei\)](#).
- Fix bug with event subscription in DDLWorker which rarely may lead to query hangs in ON CLUSTER. Introduced in [#13450](#). [#15477 \(alesapin\)](#).
- Report proper error when the second argument of boundingRatio aggregate function has a wrong type. [#15407 \(detailyang\)](#).
- Fixes [#15365](#): attach a database with MySQL engine throws exception (no query context). [#15384 \(Winter Zhang\)](#).
- Fix the case of multiple occurrences of column transformers in a select query. [#15378 \(Amos Bird\)](#).
- Fixed compression in S3 storage. [#15376 \(Vladimir Chebotarev\)](#).
- Fix bug where queries like SELECT toStartOfDay(today()) fail complaining about empty time\_zone argument. [#15319 \(Bharat Nallan\)](#).
- Fix race condition during MergeTree table rename and background cleanup. [#15304 \(alesapin\)](#).
- Fix rare race condition on server startup when system logs are enabled. [#15300 \(alesapin\)](#).
- Fix hang of queries with a lot of subqueries to same table of MySQL engine. Previously, if there were more than 16 subqueries to same MySQL table in query, it hang forever. [#15299 \(Anton Popov\)](#).
- Fix MSan report in QueryLog. Uninitialized memory can be used for the field memory\_usage. [#15258 \(alexey-milovidov\)](#).
- Fix 'Unknown identifier' in GROUP BY when query has JOIN over Merge table. [#15242 \(Artem Zuikov\)](#).
- Fix instance crash when using joinGet with LowCardinality types. This fixes [#15214](#). [#15220 \(Amos Bird\)](#).
- Fix bug in table engine Buffer which does not allow to insert data of new structure into Buffer after ALTER query. Fixes [#15117](#). [#15192 \(alesapin\)](#).
- Adjust Decimal field size in MySQL column definition packet. [#15152 \(maqroll\)](#).
- Fixes Data compressed with different methods in join\_algorithm='auto'. Keep LowCardinality as type for left table join key in join\_algorithm='partial\_merge'. [#15088 \(Artem Zuikov\)](#).
- Update jemalloc to fix percpu\_arena with affinity mask. [#15035 \(Azat Khuzhin\)](#). [#14957 \(Azat Khuzhin\)](#).
- We already use padded comparison between String and FixedString (<https://github.com/ClickHouse/ClickHouse/blob/master/src/Functions/FunctionsComparison.h#L333>). This PR applies the same logic to field comparison which corrects the usage of FixedString as primary keys. This fixes [#14908](#). [#15033 \(Amos Bird\)](#).

- If function `bar` was called with specifically crafted arguments, buffer overflow was possible. This closes [#13926](#). [#15028 \(alexey-milovidov\)](#).
- Fixed Cannot rename ... errno: 22, strerror: Invalid argument error on DDL query execution in Atomic database when running clickhouse-server in Docker on Mac OS. [#15024 \(tavplubix\)](#).
- Fix crash in RIGHT or FULL JOIN with `join_algorithm='auto'` when memory limit exceeded and we should change HashJoin with MergeJoin. [#15002 \(Artem Zuikov\)](#).
- Now settings `number_of_free_entries_in_pool_to_execute_mutation` and `number_of_free_entries_in_pool_to_lower_max_size_of_merge` can be equal to `background_pool_size`. [#14975 \(alesapin\)](#).
- Fix to make predicate push down work when subquery contains `finalizeAggregation` function. Fixes [#14847](#). [#14937 \(filimonov\)](#).
- Publish CPU frequencies per logical core in `system.asynchronous_metrics`. This fixes [#14923](#). [#14924 \(Alexander Kuzmenkov\)](#).
- MaterializeMySQL (experimental feature): Fixed `.metadata.tmp` File exists error. [#14898 \(Winter Zhang\)](#).
- Fix the issue when some invocations of `extractAllGroups` function may trigger "Memory limit exceeded" error. This fixes [#13383](#). [#14889 \(alexey-milovidov\)](#).
- Fix SIGSEGV for an attempt to INSERT into StorageFile with file descriptor. [#14887 \(Azat Khuzhin\)](#).
- Fixed segfault in `cache` dictionary [#14837](#). [#14879 \(Nikita Mikhaylov\)](#).
- MaterializeMySQL (experimental feature): Fixed bug in parsing MySQL binlog events, which causes `Attempt` to read after eof and `Packet payload is not fully read` in MaterializeMySQL database engine. [#14852 \(Winter Zhang\)](#).
- Fix rare error in `SELECT` queries when the queried column has `DEFAULT` expression which depends on the other column which also has `DEFAULT` and not present in select query and not exists on disk. Partially fixes [#14531](#). [#14845 \(alesapin\)](#).
- Fix a problem where the server may get stuck on startup while talking to ZooKeeper, if the configuration files have to be fetched from ZK (using the `from_zk` include option). This fixes [#14814](#). [#14843 \(Alexander Kuzmenkov\)](#).
- Fix wrong monotonicity detection for shrunk `Int -> Int` cast of signed types. It might lead to incorrect query result. This bug is unveiled in [#14513](#). [#14783 \(Amos Bird\)](#).
- Replace column transformer should replace identifiers with cloned ASTs. This fixes [#14695](#). [#14734 \(Amos Bird\)](#).
- Fixed missed default database name in metadata of materialized view when executing `ALTER ... MODIFY QUERY`. [#14664 \(tavplubix\)](#).
- Fix bug when `ALTER UPDATE` mutation with `Nullable` column in assignment expression and constant value (like `UPDATE x = 42`) leads to incorrect value in column or segfault. Fixes [#13634](#), [#14045](#). [#14646 \(alesapin\)](#).
- Fix wrong Decimal multiplication result caused wrong decimal scale of result column. [#14603 \(Artem Zuikov\)](#).
- Fix function `has` with `LowCardinality` of `Nullable`. [#14591 \(Mike\)](#).
- Cleanup data directory after Zookeeper exceptions during CreateQuery for StorageReplicatedMergeTree Engine. [#14563 \(Bharat Nallan\)](#).

- Fix rare segfaults in functions with combinator `-Resample`, which could appear in result of overflow with very large parameters. [#14562](#) ([Anton Popov](#)).
- Fix a bug when converting `Nullable(String)` to `Enum`. Introduced by [#12745](#). This fixes [#14435](#). [#14530](#) ([Amos Bird](#)).
- Fixed the incorrect sorting order of `Nullable` column. This fixes [#14344](#). [#14495](#) ([Nikita Mikhaylov](#)).
- Fix `currentDatabase()` function cannot be used in `ON CLUSTER` ddl query. [#14211](#) ([Winter Zhang](#)).
- `MaterializeMySQL` (experimental feature): Fixed `Packet payload is not fully read` error in `MaterializeMySQL` database engine. [#14696](#) ([BohuTANG](#)).

## Improvement

- Enable `Atomic` database engine by default for newly created databases. [#15003](#) ([tavplubix](#)).
- Add the ability to specify specialized codecs like `Delta`, `T64`, etc. for columns with subtypes. Implements [#12551](#), fixes [#11397](#), fixes [#4609](#). [#15089](#) ([alesapin](#)).
- Dynamic reload of zookeeper config. [#14678](#) ([sundyli](#)).
- Now it's allowed to execute `ALTER ... ON CLUSTER` queries regardless of the `<internal_replication>` setting in cluster config. [#16075](#) ([alesapin](#)).
- Now `joinGet` supports multi-key lookup. Continuation of [#12418](#). [#13015](#) ([Amos Bird](#)).
- Wait for `DROP/DETACH TABLE` to actually finish if `NO DELAY` or `SYNC` is specified for `Atomic` database. [#15448](#) ([tavplubix](#)).
- Now it's possible to change the type of version column for `VersionedCollapsingMergeTree` with `ALTER` query. [#15442](#) ([alesapin](#)).
- Unfold `{database}`, `{table}` and `{uuid}` macros in `zookeeper_path` on replicated table creation. Do not allow `RENAME TABLE` if it may break `zookeeper_path` after server restart. Fixes [#6917](#). [#15348](#) ([tavplubix](#)).
- The function `now` allows an argument with timezone. This closes [#15264](#). [#15285](#) ([flynn](#)).
- Do not allow connections to ClickHouse server until all scripts in `/docker-entrypoint-initdb.d/` are executed. [#15244](#) ([Aleksei Kozharin](#)).
- Added `optimize` setting to `EXPLAIN PLAN` query. If enabled, query plan level optimisations are applied. Enabled by default. [#15201](#) ([Nikolai Kochetov](#)).
- Proper exception message for wrong number of arguments of `CAST`. This closes [#13992](#). [#15029](#) ([alexey-milovidov](#)).
- Add option to disable TTL move on data part insert. [#15000](#) ([Pavel Kovalenko](#)).
- Ignore key constraints when doing mutations. Without this pull request, it's not possible to do mutations when `force_index_by_date = 1` or `force_primary_key = 1`. [#14973](#) ([Amos Bird](#)).
- Allow to drop Replicated table if previous drop attempt was failed due to ZooKeeper session expiration. This fixes [#11891](#). [#14926](#) ([alexey-milovidov](#)).
- Fixed excessive settings constraint violation when running `SELECT` with `SETTINGS` from a distributed table. [#14876](#) ([Amos Bird](#)).
- Provide a `load_balancing_first_offset` query setting to explicitly state what the first replica is. It's used together with `FIRST_OR_RANDOM` load balancing strategy, which allows to control replicas workload. [#14867](#) ([Amos Bird](#)).

- Show subqueries for `SET` and `JOIN` in `EXPLAIN` result. #14856 (Nikolai Kochetov).
- Allow using multi-volume storage configuration in storage `Distributed`. #14839 (Pavel Kovalenko).
- Construct `query_start_time` and `query_start_time_microseconds` from the same timespec. #14831 (Bharat Nallan).
- Support for disabling persistency for `StorageJoin` and `StorageSet`, this feature is controlled by setting `disable_set_and_join_persistency`. And this PR solved issue #6318. #14776 (vxider).
- Now `COLUMNS` can be used to wrap over a list of columns and apply column transformers afterwards. #14775 (Amos Bird).
- Add `merge_algorithm` to `system.merges` table to improve merging inspections. #14705 (Amos Bird).
- Fix potential memory leak caused by zookeeper exists watch. #14693 (hustnn).
- Allow parallel execution of distributed DDL. #14684 (Azat Khuzhin).
- Add `QueryMemoryLimitExceeded` event counter. This closes #14589. #14647 (fastio).
- Fix some trailing whitespaces in query formatting. #14595 (Azat Khuzhin).
- ClickHouse treats partition expr and key expr differently. Partition expr is used to construct an minmax index containing related columns, while primary key expr is stored as an expr. Sometimes user might partition a table at coarser levels, such as `partition by i / 1000`. However, binary operators are not monotonic and this PR tries to fix that. It might also benefit other use cases. #14513 (Amos Bird).
- Add an option to skip access checks for `DiskS3`. `s3` disk is an experimental feature. #14497 (Pavel Kovalenko).
- Speed up server shutdown process if there are ongoing S3 requests. #14496 (Pavel Kovalenko).
- `SYSTEM RELOAD CONFIG` now throws an exception if failed to reload and continues using the previous `users.xml`. The background periodic reloading also continues using the previous `users.xml` if failed to reload. #14492 (Vitaly Baranov).
- For `INSERTs` with inline data in `VALUES` format in the script mode of `clickhouse-client`, support semicolon as the data terminator, in addition to the new line. Closes #12288. #13192 (Alexander Kuzmenkov).
- Support custom codecs in compact parts. #12183 (Anton Popov).

## Performance Improvement

- Enable compact parts by default for small parts. This will allow to process frequent inserts slightly more efficiently (4..100 times). #11913 (alexey-milovidov).
- Improve `quantileTDigest` performance. This fixes #2668. #15542 (Kruglov Pavel).
- Significantly reduce memory usage in `AggregatingInOrderTransform/optimize_aggregation_in_order`. #15543 (Azat Khuzhin).
- Faster 256-bit multiplication. #15418 (Artem Zuikov).
- Improve performance of 256-bit types using `(u)int64_t` as base type for wide integers. Original wide integers use 8-bit types as base. #14859 (Artem Zuikov).
- Explicitly use a temporary disk to store vertical merge temporary data. #15639 (Grigory Pervakov).
- Use one S3 `DeleteObjects` request instead of multiple `DeleteObject` in a loop. No functionality changes, so covered by existing tests like `integration/test_log_family_s3`. #15238 (ianton-ru).

- Fix `DateTime <op> DateTime` mistakenly choosing the slow generic implementation. This fixes #15153. #15178 (Amos Bird).
- Improve performance of GROUP BY key of type `FixedString`. #15034 (Amos Bird).
- Only `mlock` code segment when starting clickhouse-server. In previous versions, all mapped regions were locked in memory, including debug info. Debug info is usually splitted to a separate file but if it isn't, it led to +2..3 GiB memory usage. #14929 (alexey-milovidov).
- ClickHouse binary become smaller due to link time optimization.

## Build/Testing/Packaging Improvement

- Now we use clang-11 for production ClickHouse build. #15239 (alesapin).
- Now we use clang-11 to build ClickHouse in CI. #14846 (alesapin).
- Switch binary builds (Linux, Darwin, AArch64, FreeDSD) to clang-11. #15622 (Ilya Yatsishin).
- Now all test images use `l1vm-symbolizer-11`. #15069 (alesapin).
- Allow to build with l1vm-11. #15366 (alexey-milovidov).
- Switch from `clang-tidy-10` to `clang-tidy-11`. #14922 (alexey-milovidov).
- Use LLVM's experimental pass manager by default. #15608 (Danila Kutenin).
- Don't allow any C++ translation unit to build more than 10 minutes or to use more than 10 GB of memory. This fixes #14925. #15060 (alexey-milovidov).
- Make performance test more stable and representative by splitting test runs and profile runs. #15027 (alexey-milovidov).
- Attempt to make performance test more reliable. It is done by remapping the executable memory of the process on the fly with `madvise` to use transparent huge pages - it can lower the number of iTLB misses which is the main source of instabilities in performance tests. #14685 (alexey-milovidov).
- Convert to python3. This closes #14886. #15007 (Azat Khuzhin).
- Fail early in functional tests if server failed to respond. This closes #15262. #15267 (alexey-milovidov).
- Allow to run AArch64 version of clickhouse-server without configs. This facilitates #15174. #15266 (alexey-milovidov).
- Improvements in CI docker images: get rid of ZooKeeper and single script for test configs installation. #15215 (alesapin).
- Fix CMake options forwarding in fast test script. Fixes error in #14711. #15155 (alesapin).
- Added a script to perform hardware benchmark in a single command. #15115 (alexey-milovidov).
- Splitted huge test `test_dictionaries_all_layouts_and_sources` into smaller ones. #15110 (Nikita Mikhaylov).
- Maybe fix MSan report in base64 (on servers with AVX-512). This fixes #14006. #15030 (alexey-milovidov).
- Reformat and cleanup code in all integration test \*.py files. #14864 (Bharat Nallan).
- Fix MaterializeMySQL empty transaction unstable test case found in CI. #14854 (Winter Zhang).
- Attempt to speed up build a little. #14808 (alexey-milovidov).
- Speed up build a little by removing unused headers. #14714 (alexey-milovidov).

- Fix build failure in OSX. #14761 (Winter Zhang).
- Enable ccache by default in cmake if it's found in OS. #14575 (alesapin).
- Control CI builds configuration from the ClickHouse repository. #14547 (alesapin).
- In CMake files: - Moved some options' descriptions' parts to comments above. - Replace 0 -> OFF, 1 -> ON in `options` default values. - Added some descriptions and links to docs to the options. - Replaced FUZZER option (there is another option `ENABLE_FUZZING` which also enables same functionality). - Removed `ENABLE_GTEST_LIBRARY` option as there is `ENABLE_TESTS`. See the full description in PR: #14711 (Mike).
- Make binary a bit smaller (~50 Mb for debug version). #14555 (Artem Zuikov).
- Use `std::filesystem::path` in `ConfigProcessor` for concatenating file paths. #14558 (Bharat Nallan).
- Fix debug assertion in `bitShiftLeft()` when called with negative big integer. #14697 (Artem Zuikov).

## ClickHouse release 20.9

### ClickHouse release v20.9.7.11-stable, 2020-12-07

#### Performance Improvement

- Fix performance of reading from `Merge` tables over huge number of `MergeTree` tables. Fixes #7748. #16988 (Anton Popov).

#### Bug Fix

- Do not restore parts from WAL if `in_memory_parts_enable_wal` is disabled. #17802 (detailyang).
- Fixed segfault when there is not enough space when inserting into `Distributed` table. #17737 (tavplubix).
- Fixed problem when ClickHouse fails to resume connection to MySQL servers. #17681 (Alexander Kazakov).
- Fixed Function not implemented error when executing `RENAME` query in Atomic database with ClickHouse running on Windows Subsystem for Linux. Fixes #17661. #17664 (tavplubix).
- When `clickhouse-client` is used in interactive mode with multiline queries, single line comment was erroneously extended till the end of query. This fixes #13654. #17565 (alexey-milovidov).
- Fix the issue when server can stop accepting connections in very rare cases. #17542 (alexey-milovidov).
- Fix alter query hang when the corresponding mutation was killed on the different replica. Fixes #16953. #17499 (alesapin).
- Fix bug when mark cache size was underestimated by `clickhouse`. It may happen when there are a lot of tiny files with marks. #17496 (alesapin).
- Fix `ORDER BY` with enabled setting `optimize_redundant_functions_in_order_by`. #17471 (Anton Popov).
- Fix duplicates after `DISTINCT` which were possible because of incorrect optimization. Fixes #17294. #17296 (li chengxiang). #17439 (Nikolai Kochetov).
- Fix crash while reading from `JOIN` table with `LowCardinality` types. Fixes #17228. #17397 (Nikolai Kochetov).
- Fix set index invalidation when there are `const` columns in the subquery. This fixes #17246 . #17249 (Amos Bird).
- Fix `ColumnConst` comparison which leads to crash. This fixed #17088 . #17135 (Amos Bird).

- Fixed crash on `CREATE TABLE ... AS some_table` query when `some_table` was created `AS table_function()`. Fixes #16944. #17072 (tavplubix).
- Bug fix for funciton fuzzBits, related issue: #16980. #17051 (hexiaoting).
- Avoid unnecessary network errors for remote queries which may be cancelled while execution, like queries with `LIMIT`. #17006 (Azat Khuzhin).
- TODO. #16866 (tavplubix).
- Return number of affected rows for `INSERT` queries via MySQL protocol. Previously ClickHouse used to always return 0, it's fixed. Fixes #16605. #16715 (Winter Zhang).

## Build/Testing/Packaging Improvement

- Update embedded timezone data to version 2020d (also update cctz to the latest master). #17204 (filimonov).

# ClickHouse release v20.9.6.14-stable, 2020-11-20

## Improvement

- Make it possible to connect to `clickhouse-server` secure endpoint which requires SNI. This is possible when `clickhouse-server` is hosted behind TLS proxy. #16938 (filimonov).
- Conditional aggregate functions (for example: `avgIf`, `sumIf`, `maxIf`) should return `NULL` when miss rows and use nullable arguments. #13964 (Winter Zhang).

## Bug Fix

- Fix bug when `ON CLUSTER` queries may hang forever for non-leader ReplicatedMergeTreeTables. #17089 (alesapin).
- Reresolve the IP of the `format_avro_schema_registry_url` in case of errors. #16985 (filimonov).
- Fix possible server crash after `ALTER TABLE ... MODIFY COLUMN ... NewType` when `SELECT` have `WHERE` expression on altering column and alter does not finished yet. #16968 (Amos Bird).
- Install script should always create subdirs in config folders. This is only relevant for Docker build with custom config. #16936 (filimonov).
- Fix possible error `Illegal type of argument` for queries with `ORDER BY`. Fixes #16580. #16928 (Nikolai Kochetov).
- Abort multipart upload if no data was written to `WriteBufferFromS3`. #16840 (Pavel Kovalenko).
- Fix crash when using `any` without any arguments. This is for #16803 . cc @azat. #16826 (Amos Bird).
- Fix `IN` operator over several columns and tuples with enabled `transform_null_in` setting. Fixes #15310. #16722 (Anton Popov).
- This will fix `optimize_read_in_order`/`optimize_aggregation_in_order` with `max_threads>0` and expression in `ORDER BY`. #16637 (Azat Khuzhin).
- fixes #16574 fixes #16231 fix remote query failure when using 'if' suffix aggregate function. #16610 (Winter Zhang).
- Query is finished faster in case of exception. Cancel execution on remote replicas if exception happens. #15578 (Azat Khuzhin).

# ClickHouse release v20.9.5.5-stable, 2020-11-13

## Bug Fix

- Fix rare silent crashes when query profiler is on and ClickHouse is installed on OS with glibc version that has (supposedly) broken asynchronous unwind tables for some functions. This fixes #15301. This fixes #13098. #16846 (alexey-milovidov).
- Now when parsing AVRO from input the LowCardinality is removed from type. Fixes #16188. #16521 (Mike).
- Fix rapid growth of metadata when using MySQL Master -> MySQL Slave -> ClickHouse MaterializeMySQL Engine, and slave\_parallel\_worker enabled on MySQL Slave, by properly shrinking GTID sets. This fixes #15951. #16504 (TCeason).
- Fix DROP TABLE for Distributed (racy with INSERT). #16409 (Azat Khuzhin).
- Fix processing of very large entries in replication queue. Very large entries may appear in ALTER queries if table structure is extremely large (near 1 MB). This fixes #16307. #16332 (alexey-milovidov).
- Fixed the inconsistent behaviour when a part of return data could be dropped because the set for its filtration wasn't created. #16308 (Nikita Mikhaylov).
- Fix bug with MySQL database. When MySQL server used as database engine is down some queries raise Exception, because they try to get tables from disabled server, while it's unnecessary. For example, query SELECT ... FROM system.parts should work only with MergeTree tables and don't touch MySQL database at all. #16032 (Kruglov Pavel).

## ClickHouse release v20.9.4.76-stable (2020-10-29)

### Bug Fix

- Fix double free in case of exception in function dictGet. It could have happened if dictionary was loaded with error. #16429 (Nikolai Kochetov).
- Fix group by with totals/rollup/cube modifiers and min/max functions over group by keys. Fixes #16393. #16397 (Anton Popov).
- Fix async Distributed INSERT w/ prefer\_localhost\_replica=0 and internal\_replication. #16358 (Azat Khuzhin).
- Fix a very wrong code in TwoLevelStringHashTable implementation, which might lead to memory leak. I'm surprised how this bug can lurk for so long.... #16264 (Amos Bird).
- Fix the case when memory can be overallocated regardless to the limit. This closes #14560. #16206 (alexey-milovidov).
- Fix ALTER MODIFY ... ORDER BY query hang for ReplicatedVersionedCollapsingMergeTree. This fixes #15980. #16011 (alesapin).
- Fix collate name & charset name parser and support length = 0 for string type. #16008 (Winter Zhang).
- Allow to use direct layout for dictionaries with complex keys. #16007 (Anton Popov).
- Prevent replica hang for 5-10 mins when replication error happens after a period of inactivity. #15987 (filimonov).
- Fix rare segfaults when inserting into or selecting from MaterializedView and concurrently dropping target table (for Atomic database engine). #15984 (tavplubix).
- Fix ambiguity in parsing of settings profiles: CREATE USER ... SETTINGS profile readonly is now considered as using a profile named readonly, not a setting named profile with the readonly constraint. This fixes #15628. #15982 (Vitaly Baranov).

- Fix a crash when database creation fails. #15954 (Winter Zhang).
- Fixed `DROP TABLE IF EXISTS` failure with `Table ... does not exist` error when table is concurrently renamed (for Atomic database engine). Fixed rare deadlock when concurrently executing some DDL queries with multiple tables (like `DROP DATABASE` and `RENAME TABLE`) Fixed `DROP/DETACH DATABASE` failure with `Table ... does not exist` when concurrently executing `DROP/DETACH TABLE`. #15934 (tavplubix).
- Fix incorrect empty result for query from Distributed table if query has `WHERE`, `PREWHERE` and `GLOBAL IN`. Fixes #15792. #15933 (Nikolai Kochetov).
- Fix possible deadlocks in RBAC. #15875 (Vitaly Baranov).
- Fix exception Block structure mismatch in `SELECT ... ORDER BY DESC` queries which were executed after `ALTER MODIFY COLUMN` query. Fixes #15800. #15852 (alesapin).
- Fix `select count()` inaccuracy for MaterializeMySQL. #15767 (tavplubix).
- Fix some cases of queries, in which only virtual columns are selected. Previously `Not found column _nothing in block` exception may be thrown. Fixes #12298. #15756 (Anton Popov).
- Fixed too low default value of `max_replicated_logs_to_keep` setting, which might cause replicas to become lost too often. Improve lost replica recovery process by choosing the most up-to-date replica to clone. Also do not remove old parts from lost replica, detach them instead. #15701 (tavplubix).
- Fix error `Cannot add simple transform to empty Pipe` which happened while reading from `Buffer` table which has different structure than destination table. It was possible if destination table returned empty result for query. Fixes #15529. #15662 (Nikolai Kochetov).
- Fixed bug with globs in S3 table function, region from URL was not applied to S3 client configuration. #15646 (Vladimir Chebotarev).
- Decrement the `ReadOnlyReplica` metric when detaching read-only tables. This fixes #15598. #15592 (sundyli).
- Throw an error when a single parameter is passed to `ReplicatedMergeTree` instead of ignoring it. #15516 (nvartolomei).

## Improvement

- Now it's allowed to execute `ALTER ... ON CLUSTER` queries regardless of the `<internal_replication>` setting in cluster config. #16075 (alesapin).
- Unfold `{database}`, `{table}` and `{uuid}` macros in `ReplicatedMergeTree` arguments on table creation. #16160 (tavplubix).

## ClickHouse release v20.9.3.45-stable (2020-10-09)

### Bug Fix

- Fix error `Cannot find column` which may happen at insertion into `MATERIALIZED VIEW` in case if query for `MV` contains `ARRAY JOIN`. #15717 (Nikolai Kochetov).
- Fix race condition in AMQP-CPP. #15667 (alesapin).
- Fix the order of destruction for resources in `ReadFromStorage` step of query plan. It might cause crashes in rare cases. Possibly connected with #15610. #15645 (Nikolai Kochetov).
- Fixed `Element ... is not a constant expression` error when using `JSON*` function result in `VALUES`, `LIMIT` or right side of `IN` operator. #15589 (tavplubix).

- Prevent the possibility of error message Could not calculate available disk space (statvfs), errno: 4, strerror: Interrupted system call. This fixes #15541. #15557 (alexey-milovidov).
- Significantly reduce memory usage in AggregatingInOrderTransform/optimize\_aggregation\_in\_order. #15543 (Azat Khuzhin).
- Mutation might hang waiting for some non-existent part after MOVE or REPLACE PARTITION or, in rare cases, after DETACH or DROP PARTITION. It's fixed. #15537 (tavplubix).
- Fix bug when ILIKE operator stops being case insensitive if LIKE with the same pattern was executed. #15536 (alesapin).
- Fix Missing columns errors when selecting columns which absent in data, but depend on other columns which also absent in data. Fixes #15530. #15532 (alesapin).
- Fix bug with event subscription in DDLWorker which rarely may lead to query hangs in ON CLUSTER. Introduced in #13450. #15477 (alesapin).
- Report proper error when the second argument of boundingRatio aggregate function has a wrong type. #15407 (detailyang).
- Fix bug where queries like SELECT toStartOfDay(today()) fail complaining about empty time\_zone argument. #15319 (Bharat Nallan).
- Fix race condition during MergeTree table rename and background cleanup. #15304 (alesapin).
- Fix rare race condition on server startup when system.logs are enabled. #15300 (alesapin).
- Fix MSan report in QueryLog. Uninitialized memory can be used for the field memory\_usage. #15258 (alexey-milovidov).
- Fix instance crash when using joinGet with LowCardinality types. This fixes #15214. #15220 (Amos Bird).
- Fix bug in table engine Buffer which does not allow to insert data of new structure into Buffer after ALTER query. Fixes #15117. #15192 (alesapin).
- Adjust decimals field size in mysql column definition packet. #15152 (maqroll).
- Fixed Cannot rename ... errno: 22, strerror: Invalid argument error on DDL query execution in Atomic database when running clickhouse-server in docker on Mac OS. #15024 (tavplubix).
- Fix to make predicate push down work when subquery contains finalizeAggregation function. Fixes #14847. #14937 (filimonov).
- Fix a problem where the server may get stuck on startup while talking to ZooKeeper, if the configuration files have to be fetched from ZK (using the from\_zk include option). This fixes #14814. #14843 (Alexander Kuzmenkov).

## Improvement

- Now it's possible to change the type of version column for VersionedCollapsingMergeTree with ALTER query. #15442 (alesapin).

## ClickHouse release v20.9.2.20, 2020-09-22

### Backward Incompatible Change

- When upgrading from versions older than 20.5, if rolling update is performed and cluster contains both versions 20.5 or greater and less than 20.5, if ClickHouse nodes with old versions are restarted and old version has been started up in presence of newer versions, it may lead to `Part ... intersects previous part` errors. To prevent this error, first install newer clickhouse-server packages on all cluster nodes and then do restarts (so, when clickhouse-server is restarted, it will start up with the new version).

## New Feature

- Added column transformers `EXCEPT`, `REPLACE`, `APPLY`, which can be applied to the list of selected columns (after `*` or `COLUMNS(...)`). For example, you can write `SELECT * EXCEPT(URL) REPLACE(number + 1 AS number)`. Another example: `select * apply(length) apply(max) from wide_string_table` to find out the maximum length of all string columns. [#14233 \(Amos Bird\)](#).
- Added an aggregate function `rankCorr` which computes a rank correlation coefficient. [#11769 \(antikvist\)](#) [#14411 \(Nikita Mikhaylov\)](#).
- Added table function `view` which turns a subquery into a table object. This helps passing queries around. For instance, it can be used in remote/cluster table functions. [#12567 \(Amos Bird\)](#).

## Bug Fix

- Fix bug when `ALTER UPDATE` mutation with `Nullable` column in assignment expression and constant value (like `UPDATE x = 42`) leads to incorrect value in column or segfault. Fixes [#13634](#), [#14045](#), [#14646 \(alesapin\)](#).
- Fix wrong Decimal multiplication result caused wrong decimal scale of result column. [#14603 \(Artem Zuikov\)](#).
- Fixed the incorrect sorting order of `Nullable` column. This fixes [#14344](#), [#14495 \(Nikita Mikhaylov\)](#).
- Fixed inconsistent comparison with primary key of type `FixedString` on index analysis if they're compared with a string of less size. This fixes [#14908](#), [#15033 \(Amos Bird\)](#).
- Fix bug which leads to wrong merges assignment if table has partitions with a single part. [#14444 \(alesapin\)](#).
- If function `bar` was called with specifically crafted arguments, buffer overflow was possible. This closes [#13926](#), [#15028 \(alexey-milovidov\)](#).
- Publish CPU frequencies per logical core in `system.asynchronous_metrics`. This fixes [#14923](#), [#14924 \(Alexander Kuzmenkov\)](#).
- Fixed `.metadata.tmp` File exists error when using `MaterializeMySQL` database engine. [#14898 \(Winter Zhang\)](#).
- Fix the issue when some invocations of `extractAllGroups` function may trigger "Memory limit exceeded" error. This fixes [#13383](#), [#14889 \(alexey-milovidov\)](#).
- Fix `SIGSEGV` for an attempt to `INSERT` into `StorageFile(fd)`. [#14887 \(Azat Khuzhin\)](#).
- Fix rare error in `SELECT` queries when the queried column has `DEFAULT` expression which depends on the other column which also has `DEFAULT` and not present in select query and not exists on disk. Partially fixes [#14531](#), [#14845 \(alesapin\)](#).
- Fix wrong monotonicity detection for shrunk `Int -> Int` cast of signed types. It might lead to incorrect query result. This bug is unveiled in [#14513](#), [#14783 \(Amos Bird\)](#).
- Fixed missed default database name in metadata of materialized view when executing `ALTER ... MODIFY QUERY`. [#14664 \(tavplubix\)](#).
- Fix possibly incorrect result of function `has` when LowCardinality and `Nullable` types are involved. [#14591 \(Mike\)](#).

- Cleanup data directory after Zookeeper exceptions during CREATE query for tables with ReplicatedMergeTree Engine. [#14563](#) (**Bharat Nallan**).
- Fix rare segfaults in functions with combinator `-Resample`, which could appear in result of overflow with very large parameters. [#14562](#) (**Anton Popov**).
- Check for array size overflow in `topK` aggregate function. Without this check the user may send a query with carefully crafted parameters that will lead to server crash. This closes [#14452](#). [#14467](#) (**alexey-milovidov**).
- Proxy restart/start/stop/reload of SysVinit to systemd (if it is used). [#14460](#) (**Azat Khuzhin**).
- Stop query execution if exception happened in `PipelineExecutor` itself. This could prevent rare possible query hung. [#14334](#) [#14402](#) (**Nikolai Kochetov**).
- Fix crash during `ALTER` query for table which was created `AS table_function`. Fixes [#14212](#). [#14326](#) (**alesapin**).
- Fix exception during `ALTER LIVE VIEW` query with `REFRESH` command. `LIVE VIEW` is an experimental feature. [#14320](#) (**Bharat Nallan**).
- Fix `QueryPlan` lifetime (for `EXPLAIN PIPELINE graph=1`) for queries with nested interpreter. [#14315](#) (**Azat Khuzhin**).
- Better check for tuple size in SSD cache complex key external dictionaries. This fixes [#13981](#). [#14313](#) (**alexey-milovidov**).
- Disallows `CODEC` on `ALIAS` column type. Fixes [#13911](#). [#14263](#) (**Bharat Nallan**).
- Fix `GRANT ALL` statement when executed on a non-global level. [#13987](#) (**Vitaly Baranov**).
- Fix `arrayJoin()` capturing in lambda (exception with logical error message was thrown). [#13792](#) (**Azat Khuzhin**).

## Experimental Feature

- Added `db-generator` tool for random database generation by given `SELECT` queries. It may facilitate reproducing issues when there is only incomplete bug report from the user. [#14442](#) (**Nikita Mikhaylov**) [#10973](#) (**ZeDRoman**).

## Improvement

- Allow using multi-volume storage configuration in storage Distributed. [#14839](#) (**Pavel Kovalenko**).
- Disallow empty `time_zone` argument in `toStartOf*` type of functions. [#14509](#) (**Bharat Nallan**).
- MySQL handler returns `OK` for queries like `SET @@var = value`. Such statement is ignored. It is needed because some MySQL drivers send `SET @@` query for setup after handshake  
<https://github.com/ClickHouse/ClickHouse/issues/9336#issuecomment-686222422> . [#14469](#) (**BohuTANG**).
- Now TTLs will be applied during merge if they were not previously materialized. [#14438](#) (**alesapin**).
- Now `clickhouse-obfuscator` supports `UUID` type as proposed in [#13163](#). [#14409](#) (**dimarub2000**).
- Added new setting `system_events_show_zero_values` as proposed in [#11384](#). [#14404](#) (**dimarub2000**).
- Implicitly convert primary key to not null in `MaterializeMySQL` (Same as MySQL). Fixes [#14114](#). [#14397](#) (**Winter Zhang**).
- Replace wide integers (256 bit) from boost multiprecision with implementation from <https://github.com/cerevra/int>. 256bit integers are experimental. [#14229](#) (**Artem Zuikov**).

- Add default compression codec for parts in `system.part_log` with the name `default_compression_codec`. #14116 (alesapin).
- Add precision argument for `DateTime` type. It allows to use `DateTime` name instead of `DateTime64`. #13761 (Winter Zhang).
- Added requirepass authorization for `Redis` external dictionary. #13688 (Ivan Torgashov).
- Improvements in RabbitMQ engine: added connection and channels failure handling, proper commits, insert failures handling, better exchanges, queue durability and queue resume opportunity, new queue settings. Fixed tests. #12761 (Kseniia Sumarokova).
- Support custom codecs in compact parts. #12183 (Anton Popov).

## Performance Improvement

- Optimize queries with `LIMIT/LIMIT BY/ORDER BY` for distributed with `GROUP BY sharding_key` (under `optimize_skip_unused_shards` and `optimize_distributed_group_by_sharding_key`). #10373 (Azat Khuzhin).
- Creating sets for multiple `JOIN` and `IN` in parallel. It may slightly improve performance for queries with several different `IN` subquery expressions. #14412 (Nikolai Kochetov).
- Improve Kafka engine performance by providing independent thread for each consumer. Separate thread pool for streaming engines (like Kafka). #13939 (fastio).

## Build/Testing/Packaging Improvement

- Lower binary size in debug build by removing debug info from `Functions`. This is needed only for one internal project in Yandex who is using very old linker. #14549 (alexey-milovidov).
- Prepare for build with clang 11. #14455 (alexey-milovidov).
- Fix the logic in backport script. In previous versions it was triggered for any labels of 100% red color. It was strange. #14433 (alexey-milovidov).
- Integration tests use default base config. All config changes are explicit with `main_configs`, `user_configs` and `dictionaries` parameters for instance. #13647 (Ilya Yatsishin).

# ClickHouse release 20.8

## ClickHouse release v20.8.12.2-lts, 2021-01-16

### Bug Fix

- Fix \*If combinator with unary function and Nullable types. #18806 (Azat Khuzhin).
- Restrict merges from wide to compact parts. In case of vertical merge it led to broken result part. #18381 (Anton Popov).

## ClickHouse release v20.8.11.17-lts, 2020-12-25

### Bug Fix

- Disable write with AIO during merges because it can lead to extremely rare data corruption of primary key columns during merge. #18481 (alesapin).
- Fixed `value is too short` error when executing `toType(...)` functions (`toDate`, `toUInt32`, etc) with argument of type `Nullable(String)`. Now such functions return `NULL` on parsing errors instead of throwing exception. Fixes #7673. #18445 (tavplubix).
- Fix possible crashes in aggregate functions with combinator `Distinct`, while using two-level aggregation. Fixes #17682. #18365 (Anton Popov).

# ClickHouse release v20.8.10.13-Its, 2020-12-24

## Bug Fix

- When server log rotation was configured using `logger.size` parameter with numeric value larger than  $2^{32}$ , the logs were not rotated properly. [#17905](#) ([Alexander Kuzmenkov](#)).
- Fixed incorrect initialization of `max_compress_block_size` in `MergeTreeWriterSettings` with `min_compress_block_size`. [#17833](#) ([flynn](#)).
- Fixed problem when ClickHouse fails to resume connection to MySQL servers. [#17681](#) ([Alexander Kazakov](#)).
- Fixed `ALTER` query hang when the corresponding mutation was killed on the different replica. This fixes [#16953](#). [#17499](#) ([alesapin](#)).
- Fixed a bug when mark cache size was underestimated by ClickHouse. It may happen when there are a lot of tiny files with marks. [#17496](#) ([alesapin](#)).
- Fixed `ORDER BY` with enabled setting `optimize_redundant_functions_in_order_by`. [#17471](#) ([Anton Popov](#)).
- Fixed `ColumnConst` comparison which leads to crash. This fixed [#17088](#) . [#17135](#) ([Amos Bird](#)).
- Fixed bug when `ON CLUSTER` queries may hang forever for non-leader ReplicatedMergeTreeTables. [#17089](#) ([alesapin](#)).
- Avoid unnecessary network errors for remote queries which may be cancelled while execution, like queries with `LIMIT`. [#17006](#) ([Azat Khuzhin](#)).
- Reresolve the IP of the `format_avro_schema_registry_url` in case of errors. [#16985](#) ([filimonov](#)).
- Fixed possible server crash after `ALTER TABLE ... MODIFY COLUMN ... NewType` when `SELECT` have `WHERE` expression on altering column and alter does not finished yet. [#16968](#) ([Amos Bird](#)).
- Install script should always create subdirs in config folders. This is only relevant for Docker build with custom config. [#16936](#) ([filimonov](#)).
- Fixed possible error `Illegal type of argument` for queries with `ORDER BY`. Fixes [#16580](#). [#16928](#) ([Nikolai Kochetov](#)).
- Abort multipart upload if no data was written to `WriteBufferFromS3`. [#16840](#) ([Pavel Kovalenko](#)).
- Fixed crash when using `any` without any arguments. This fixes [#16803](#). [#16826](#) ([Amos Bird](#)).
- Fixed `IN` operator over several columns and tuples with enabled `transform_null_in` setting. Fixes [#15310](#). [#16722](#) ([Anton Popov](#)).
- Fixed inconsistent behaviour of `optimize_read_in_order/optimize_aggregation_in_order` with `max_threads > 0` and expression in `ORDER BY`. [#16637](#) ([Azat Khuzhin](#)).
- Fixed the issue when query optimization was producing wrong result if query contains `ARRAY JOIN`. [#17887](#) ([sundylj](#)).
- Query is finished faster in case of exception. Cancel execution on remote replicas if exception happens. [#15578](#) ([Azat Khuzhin](#)).

# ClickHouse release v20.8.6.6-Its, 2020-11-13

## Bug Fix

- Fix rare silent crashes when query profiler is on and ClickHouse is installed on OS with glibc version that has (supposedly) broken asynchronous unwind tables for some functions. This fixes #15301. This fixes #13098. #16846 (alexey-milovidov).
- Now when parsing AVRO from input the LowCardinality is removed from type. Fixes #16188. #16521 (Mike).
- Fix rapid growth of metadata when using MySQL Master -> MySQL Slave -> ClickHouse MaterializeMySQL Engine, and slave\_parallel\_worker enabled on MySQL Slave, by properly shrinking GTID sets. This fixes #15951. #16504 (TCeason).
- Fix DROP TABLE for Distributed (racy with INSERT). #16409 (Azat Khuzhin).
- Fix processing of very large entries in replication queue. Very large entries may appear in ALTER queries if table structure is extremely large (near 1 MB). This fixes #16307. #16332 (alexey-milovidov).
- Fixed the inconsistent behaviour when a part of return data could be dropped because the set for its filtration wasn't created. #16308 (Nikita Mikhaylov).
- Fix bug with MySQL database. When MySQL server used as database engine is down some queries raise Exception, because they try to get tables from disabled server, while it's unnecessary. For example, query SELECT ... FROM system.parts should work only with MergeTree tables and don't touch MySQL database at all. #16032 (Kruglov Pavel).

## ClickHouse release v20.8.5.45-lts, 2020-10-29

### Bug Fix

- Fix double free in case of exception in function dictGet. It could have happened if dictionary was loaded with error. #16429 (Nikolai Kochetov).
- Fix group by with totals/rollup/cube modifiers and min/max functions over group by keys. Fixes #16393. #16397 (Anton Popov).
- Fix async Distributed INSERT w/ prefer\_localhost\_replica=0 and internal\_replication. #16358 (Azat Khuzhin).
- Fix a possible memory leak during GROUP BY with string keys, caused by an error in TwoLevelStringHashTable implementation. #16264 (Amos Bird).
- Fix the case when memory can be overallocated regardless to the limit. This closes #14560. #16206 (alexey-milovidov).
- Fix ALTER MODIFY ... ORDER BY query hang for ReplicatedVersionedCollapsingMergeTree. This fixes #15980. #16011 (alesapin).
- Fix collate name & charset name parser and support length = 0 for string type. #16008 (Winter Zhang).
- Allow to use direct layout for dictionaries with complex keys. #16007 (Anton Popov).
- Prevent replica hang for 5-10 mins when replication error happens after a period of inactivity. #15987 (filimonov).
- Fix rare segfaults when inserting into or selecting from MaterializedView and concurrently dropping target table (for Atomic database engine). #15984 (tavplubix).
- Fix ambiguity in parsing of settings profiles: CREATE USER ... SETTINGS profile readonly is now considered as using a profile named readonly, not a setting named profile with the readonly constraint. This fixes #15628. #15982 (Vitaly Baranov).
- Fix a crash when database creation fails. #15954 (Winter Zhang).

- Fixed `DROP TABLE IF EXISTS` failure with `Table ... does not exist` error when table is concurrently renamed (for Atomic database engine). Fixed rare deadlock when concurrently executing some DDL queries with multiple tables (like `DROP DATABASE` and `RENAME TABLE`) Fixed `DROP/DETACH DATABASE` failure with `Table ... does not exist` when concurrently executing `DROP/DETACH TABLE`. #15934 (tavplubix).
- Fix incorrect empty result for query from Distributed table if query has `WHERE`, `PREWHERE` and `GLOBAL IN`. Fixes #15792. #15933 (Nikolai Kochetov).
- Fix possible deadlocks in RBAC. #15875 (Vitaly Baranov).
- Fix exception `Block structure mismatch` in `SELECT ... ORDER BY DESC` queries which were executed after `ALTER MODIFY COLUMN` query. Fixes #15800. #15852 (alesapin).
- Fix some cases of queries, in which only virtual columns are selected. Previously `Not found column _nothing in block` exception may be thrown. Fixes #12298. #15756 (Anton Popov).
- Fix error `Cannot find column` which may happen at insertion into `MATERIALIZED VIEW` in case if query for `MV` contains `ARRAY JOIN`. #15717 (Nikolai Kochetov).
- Fixed too low default value of `max_replicated_logs_to_keep` setting, which might cause replicas to become lost too often. Improve lost replica recovery process by choosing the most up-to-date replica to clone. Also do not remove old parts from lost replica, detach them instead. #15701 (tavplubix).
- Fix error `Cannot add simple transform to empty Pipe` which happened while reading from `Buffer` table which has different structure than destination table. It was possible if destination table returned empty result for query. Fixes #15529. #15662 (Nikolai Kochetov).
- Fixed bug with globs in S3 table function, region from URL was not applied to S3 client configuration. #15646 (Vladimir Chebotarev).
- Decrement the `ReadOnlyReplica` metric when detaching read-only tables. This fixes #15598. #15592 (sundyl).
- Throw an error when a single parameter is passed to `ReplicatedMergeTree` instead of ignoring it. #15516 (nvartolomei).

## Improvement

- Now it's allowed to execute `ALTER ... ON CLUSTER` queries regardless of the `<internal_replication>` setting in cluster config. #16075 (alesapin).
- Unfold `{database}`, `{table}` and `{uuid}` macros in `ReplicatedMergeTree` arguments on table creation. #16159 (tavplubix).

## ClickHouse release v20.8.4.11-lts, 2020-10-09

### Bug Fix

- Fix the order of destruction for resources in `ReadFromStorage` step of query plan. It might cause crashes in rare cases. Possibly connected with #15610. #15645 (Nikolai Kochetov).
- Fixed `Element ... is not a constant expression` error when using `JSON*` function result in `VALUES`, `LIMIT` or right side of `IN` operator. #15589 (tavplubix).
- Prevent the possibility of error message `Could not calculate available disk space (statvfs), errno: 4, strerror: Interrupted system call`. This fixes #15541. #15557 (alexey-milovidov).
- Significantly reduce memory usage in `AggregatingInOrderTransform/optimize_aggregation_in_order`. #15543 (Azat Khuzhin).

- Mutation might hang waiting for some non-existent part after `MOVE` or `REPLACE PARTITION` or, in rare cases, after `DETACH` or `DROP PARTITION`. It's fixed. [#15537 \(tavplubix\)](#).
- Fix bug when `ILIKE` operator stops being case insensitive if `LIKE` with the same pattern was executed. [#15536 \(alesapin\)](#).
- Fix `Missing columns` errors when selecting columns which absent in data, but depend on other columns which also absent in data. Fixes [#15530](#). [#15532 \(alesapin\)](#).
- Fix bug with event subscription in `DDLWorker` which rarely may lead to query hangs in `ON CLUSTER`. Introduced in [#13450](#). [#15477 \(alesapin\)](#).
- Report proper error when the second argument of `boundingRatio` aggregate function has a wrong type. [#15407 \(detailyang\)](#).
- Fix race condition during `MergeTree` table rename and background cleanup. [#15304 \(alesapin\)](#).
- Fix rare race condition on server startup when `system.logs` are enabled. [#15300 \(alesapin\)](#).
- Fix MSan report in `QueryLog`. Uninitialized memory can be used for the field `memory_usage`. [#15258 \(alexey-milovidov\)](#).
- Fix instance crash when using `joinGet` with `LowCardinality` types. This fixes [#15214](#). [#15220 \(Amos Bird\)](#).
- Fix bug in table engine `Buffer` which does not allow to insert data of new structure into `Buffer` after `ALTER` query. Fixes [#15117](#). [#15192 \(alesapin\)](#).
- Adjust decimals field size in `mysql` column definition packet. [#15152 \(maqroll\)](#).
- We already use padded comparison between `String` and `FixedString` (<https://github.com/ClickHouse/ClickHouse/blob/master/src/Functions/FunctionsComparison.h#L333>). This PR applies the same logic to field comparison which corrects the usage of `FixedString` as primary keys. This fixes [#14908](#). [#15033 \(Amos Bird\)](#).
- If function `bar` was called with specifically crafted arguments, buffer overflow was possible. This closes [#13926](#). [#15028 \(alexey-milovidov\)](#).
- Fixed Cannot rename ... `errno: 22, strerror: Invalid argument` error on DDL query execution in Atomic database when running `clickhouse-server` in docker on Mac OS. [#15024 \(tavplubix\)](#).
- Now settings `number_of_free_entries_in_pool_to_execute_mutation` and `number_of_free_entries_in_pool_to_lower_max_size_of_merge` can be equal to `background_pool_size`. [#14975 \(alesapin\)](#).
- Fix to make predicate push down work when subquery contains `finalizeAggregation` function. Fixes [#14847](#). [#14937 \(filimonov\)](#).
- Publish CPU frequencies per logical core in `system.asynchronous_metrics`. This fixes [#14923](#). [#14924 \(Alexander Kuzmenkov\)](#).
- Fixed `.metadata.tmp` File exists error when using `MaterializeMySQL` database engine. [#14898 \(Winter Zhang\)](#).
- Fix a problem where the server may get stuck on startup while talking to ZooKeeper, if the configuration files have to be fetched from ZK (using the `from_zk` include option). This fixes [#14814](#). [#14843 \(Alexander Kuzmenkov\)](#).
- Fix wrong monotonicity detection for shrunk `Int -> Int` cast of signed types. It might lead to incorrect query result. This bug is unveiled in [#14513](#). [#14783 \(Amos Bird\)](#).
- Fixed the incorrect sorting order of `Nullable` column. This fixes [#14344](#). [#14495 \(Nikita Mikhaylov\)](#).

## Improvement

- Now it's possible to change the type of version column for VersionedCollapsingMergeTree with ALTER query. #15442 (alesapin).

## ClickHouse release v20.8.3.18-stable, 2020-09-18

### Bug Fix

- Fix the issue when some invocations of extractAllGroups function may trigger "Memory limit exceeded" error. This fixes #13383. #14889 (alexey-milovidov).
- Fix SIGSEGV for an attempt to INSERT into StorageFile(fd). #14887 (Azat Khuzhin).
- Fix rare error in SELECT queries when the queried column has DEFAULT expression which depends on the other column which also has DEFAULT and not present in select query and not exists on disk. Partially fixes #14531. #14845 (alesapin).
- Fixed missed default database name in metadata of materialized view when executing ALTER ... MODIFY QUERY. #14664 (tavplubix).
- Fix bug when ALTER UPDATE mutation with Nullable column in assignment expression and constant value (like UPDATE x = 42) leads to incorrect value in column or segfault. Fixes #13634, #14045. #14646 (alesapin).
- Fix wrong Decimal multiplication result caused wrong decimal scale of result column. #14603 (Artem Zuikov).
- Added the checker as neither calling Ic->isNullable() nor calling Is->getDictionaryPtr()->isNullable() would return the correct result. #14591 (myrrc).
- Cleanup data directory after Zookeeper exceptions during CreateQuery for StorageReplicatedMergeTree Engine. #14563 (Bharat Nallan).
- Fix rare segfaults in functions with combinator -Resample, which could appear in result of overflow with very large parameters. #14562 (Anton Popov).

## Improvement

- Speed up server shutdown process if there are ongoing S3 requests. #14858 (Pavel Kovalenko).
- Allow using multi-volume storage configuration in storage Distributed. #14839 (Pavel Kovalenko).
- Speed up server shutdown process if there are ongoing S3 requests. #14496 (Pavel Kovalenko).
- Support custom codecs in compact parts. #12183 (Anton Popov).

## ClickHouse release v20.8.2.3-stable, 2020-09-08

### Backward Incompatible Change

- Now OPTIMIZE FINAL query does not recalculate TTL for parts that were added before TTL was created. Use ALTER TABLE ... MATERIALIZE TTL once to calculate them, after that OPTIMIZE FINAL will evaluate TTL's properly. This behavior never worked for replicated tables. #14220 (alesapin).
- Extend parallel\_distributed\_insert\_select setting, adding an option to run INSERT into local table. The setting changes type from Bool to UInt64, so the values false and true are no longer supported. If you have these values in server configuration, the server will not start. Please replace them with 0 and 1, respectively. #14060 (Azat Khuzhin).

- Remove support for the `ODBCDriver` input/output format. This was a deprecated format once used for communication with the ClickHouse ODBC driver, now long superseded by the `ODBCDriver2` format. Resolves #13629. #13847 (hexiaoting).
- When upgrading from versions older than 20.5, if rolling update is performed and cluster contains both versions 20.5 or greater and less than 20.5, if ClickHouse nodes with old versions are restarted and old version has been started up in presence of newer versions, it may lead to `Part ... intersects previous part` errors. To prevent this error, first install newer `clickhouse-server` packages on all cluster nodes and then do restarts (so, when `clickhouse-server` is restarted, it will start up with the new version).

## New Feature

- Add the ability to specify `Default` compression codec for columns that correspond to settings specified in `config.xml`. Implements: #9074. #14049 (alesapin).
- Support Kerberos authentication in Kafka, using `krb5` and `cyrus-sasl` libraries. #12771 (Ilya Golshtain).
- Add function `normalizeQuery` that replaces literals, sequences of literals and complex aliases with placeholders. Add function `normalizedQueryHash` that returns identical 64bit hash values for similar queries. It helps to analyze query log. This closes #11271. #13816 (alexey-milovidov).
- Add `time_zones` table. #13880 (Bharat Nallan).
- Add function `defaultValueOfTypeName` that returns the default value for a given type. #13877 (hcz).
- Add `countDigits(x)` function that count number of decimal digits in integer or decimal column. Add `isDecimalOverflow(d, [p])` function that checks if the value in Decimal column is out of its (or specified) precision. #14151 (Artem Zuikov).
- Add `quantileExactLow` and `quantileExactHigh` implementations with respective aliases for `medianExactLow` and `medianExactHigh`. #13818 (Bharat Nallan).
- Added `date_trunc` function that truncates a date/time value to a specified date/time part. #13888 (Vladimir Golovchenko).
- Add new optional section `<user_directories>` to the main config. #13425 (Vitaly Baranov).
- Add `ALTER SAMPLE BY` statement that allows to change table sample clause. #13280 (Amos Bird).
- Function `position` now supports optional `start_pos` argument. #13237 (vdimir).

## Bug Fix

- Fix visible data clobbering by progress bar in client in interactive mode. This fixes #12562 and #13369 and #13584 and fixes #12964. #13691 (alexey-milovidov).
- Fixed incorrect sorting order if `LowCardinality` column when sorting by multiple columns. This fixes #13958. #14223 (Nikita Mikhaylov).
- Check for array size overflow in `topK` aggregate function. Without this check the user may send a query with carefully crafted parameters that will lead to server crash. This closes #14452. #14467 (alexey-milovidov).
- Fix bug which can lead to wrong merges assignment if table has partitions with a single part. #14444 (alesapin).
- Stop query execution if exception happened in `PipelineExecutor` itself. This could prevent rare possible query hung. Continuation of #14334. #14402 #14334 (Nikolai Kochetov).
- Fix crash during `ALTER` query for table which was created `AS table_function`. Fixes #14212. #14326 (alesapin).

- Fix exception during ALTER LIVE VIEW query with REFRESH command. Live view is an experimental feature. #14320 (Bharat Nallan).
- Fix QueryPlan lifetime (for EXPLAIN PIPELINE graph=1) for queries with nested interpreter. #14315 (Azat Khuzhin).
- Fix segfault in `clickhouse-odbc-bridge` during schema fetch from some external sources. This PR fixes #13861. #14267 (Vitaly Baranov).
- Fix crash in mark inclusion search introduced in #12277. #14225 (Amos Bird).
- Fix creation of tables with named tuples. This fixes #13027. #14143 (alexey-milovidov).
- Fix formatting of minimal negative decimal numbers. This fixes #14111. #14119 (Alexander Kuzmenkov).
- Fix `DistributedFilesToInsert` metric (zeroed when it should not). #14095 (Azat Khuzhin).
- Fix `pointInPolygon` with const 2d array as polygon. #14079 (Alexey Ilyukhov).
- Fixed wrong mount point in extra info for `Poco::Exception`: no space left on device #14050 (tavplubix).
- Fix GRANT ALL statement when executed on a non-global level. #13987 (Vitaly Baranov).
- Fix parser to reject create table as table function with engine. #13940 (hcz).
- Fix wrong results in select queries with `DISTINCT` keyword and subqueries with UNION ALL in case `optimize_duplicate_order_by_and_distinct` setting is enabled. #13925 (Artem Zuikov).
- Fixed potential deadlock when renaming `Distributed` table. #13922 (tavplubix).
- Fix incorrect sorting for `FixedString` columns when sorting by multiple columns. Fixes #13182. #13887 (Nikolai Kochetov).
- Fix potentially imprecise result of `topK/topKWeighted` merge (with non-default parameters). #13817 (Azat Khuzhin).
- Fix reading from MergeTree table with INDEX of type SET fails when comparing against NULL. This fixes #13686. #13793 (Amos Bird).
- Fix `arrayJoin` capturing in lambda (LOGICAL\_ERROR). #13792 (Azat Khuzhin).
- Add step overflow check in function `range`. #13790 (Azat Khuzhin).
- Fixed `Directory not empty` error when concurrently executing `DROP DATABASE` and `CREATE TABLE`. #13756 (alexey-milovidov).
- Add range check for `h3KRing` function. This fixes #13633. #13752 (alexey-milovidov).
- Fix race condition between DETACH and background merges. Parts may revive after detach. This is continuation of #8602 that did not fix the issue but introduced a test that started to fail in very rare cases, demonstrating the issue. #13746 (alexey-milovidov).
- Fix logging `Settings.Names/Values` when `log_queries_min_type > QUERY_START`. #13737 (Azat Khuzhin).
- Fixes `/replicas_status` endpoint response status code when `verbose=1`. #13722 (javi santana).
- Fix incorrect message in `clickhouse-server.init` while checking user and group. #13711 (ylchou).
- Do not optimize any(`arrayJoin()`) -> `arrayJoin()` under `optimize_move_functions_out_of_any` setting. #13681 (Azat Khuzhin).

- Fix crash in JOIN with StorageMerge and `set enable_optimize_predicate_expression=1`. #13679 (Artem Zuikov).
- Fix typo in error message about `The value of 'number_of_free_entries_in_pool_to_lower_max_size_of_merge'` setting. #13678 (alexey-milovidov).
- Concurrent `ALTER ... REPLACE/MOVE PARTITION ...` queries might cause deadlock. It's fixed. #13626 (tavplubix).
- Fixed the behaviour when sometimes cache-dictionary returned default value instead of present value from source. #13624 (Nikita Mikhaylov).
- Fix secondary indices corruption in compact parts. Compact parts are experimental feature. #13538 (Anton Popov).
- Fix premature `ON CLUSTER` timeouts for queries that must be executed on a single replica. Fixes #6704, #7228, #13361, #11884. #13450 (alesapin).
- Fix wrong code in function `netloc`. This fixes #13335. #13446 (alexey-milovidov).
- Fix possible race in `StorageMemory`. #13416 (Nikolai Kochetov).
- Fix missing or excessive headers in `TSV/CSVWithNames` formats in HTTP protocol. This fixes #12504. #13343 (Azat Khuzhin).
- Fix parsing row policies from `users.xml` when names of databases or tables contain dots. This fixes #5779, #12527. #13199 (Vitaly Baranov).
- Fix access to `redis` dictionary after connection was dropped once. It may happen with `cache` and `direct` dictionary layouts. #13082 (Anton Popov).
- Removed wrong auth access check when using `ClickHouseDictionarySource` to query remote tables. #12756 (sundyl).
- Properly distinguish subqueries in some cases for common subexpression elimination. #8333. #8367 (Amos Bird).

## Improvement

- Disallows `CODEC` on `ALIAS` column type. Fixes #13911. #14263 (Bharat Nallan).
- When waiting for a dictionary update to complete, use the timeout specified by `query_wait_timeout_milliseconds` setting instead of a hard-coded value. #14105 (Nikita Mikhaylov).
- Add setting `min_index_granularity_bytes` that protects against accidentally creating a table with very low `index_granularity_bytes` setting. #14139 (Bharat Nallan).
- Now it's possible to fetch partitions from clusters that use different ZooKeeper: `ALTER TABLE table_name FETCH PARTITION partition_expr FROM 'zk-name:/path-in-zookeeper'`. It's useful for shipping data to new clusters. #14155 (Amos Bird).
- Slightly better performance of Memory table if it was constructed from a huge number of very small blocks (that's unlikely). Author of the idea: [Mark Papadakis](#). Closes #14043. #14056 (alexey-milovidov).
- Conditional aggregate functions (for example: `avgIf`, `sumIf`, `maxIf`) should return `NULL` when miss rows and use nullable arguments. #13964 (Winter Zhang).
- Increase limit in -Resample combinator to 1M. #13947 (Mikhail f. Shiryaev).
- Corrected an error in AvroConfluent format that caused the Kafka table engine to stop processing messages when an abnormally small, malformed, message was received. #13941 (Gervasio Varela).

- Fix wrong error for long queries. It was possible to get syntax error other than Max query size exceeded for correct query. #13928 (Nikolai Kochetov).
- Better error message for null value of TabSeparated format. #13906 (jiang tao).
- Function arrayCompact will compare NaNs bitwise if the type of array elements is Float32/Float64. In previous versions NaNs were always not equal if the type of array elements is Float32/Float64 and were always equal if the type is more complex, like Nullable(Float64). This closes #13857. #13868 (alexey-milovidov).
- Fix data race in Igamma function. This race was caught only in tsan, no side effects a really happened. #13842 (Nikolai Kochetov).
- Avoid too slow queries when arrays are manipulated as fields. Throw exception instead. #13753 (alexey-milovidov).
- Added Redis requirepass authorization (for redis dictionary source). #13688 (Ivan Torgashov).
- Add MergeTree Write-Ahead-Log (WAL) dump tool. WAL is an experimental feature. #13640 (BohuTANG).
- In previous versions lcm function may produce assertion violation in debug build if called with specifically crafted arguments. This fixes #13368. #13510 (alexey-milovidov).
- Provide monotonicity for toDate/toDateTime functions in more cases. Monotonicity information is used for index analysis (more complex queries will be able to use index). Now the input arguments are saturated more naturally and provides better monotonicity. #13497 (Amos Bird).
- Support compound identifiers for custom settings. Custom settings is an integration point of ClickHouse codebase with other codebases (no benefits for ClickHouse itself) #13496 (Vitaly Baranov).
- Move parts from DiskLocal to DiskS3 in parallel. DiskS3 is an experimental feature. #13459 (Pavel Kovalenko).
- Enable mixed granularity parts by default. #13449 (alesapin).
- Proper remote host checking in S3 redirects (security-related thing). #13404 (Vladimir Chebotarev).
- Add QueryTimeMicroseconds, SelectQueryTimeMicroseconds and InsertQueryTimeMicroseconds to system.events. #13336 (ianton-ru).
- Fix debug assertion when Decimal has too large negative exponent. Fixes #13188. #13228 (alexey-milovidov).
- Added cache layer for DiskS3 (cache to local disk mark and index files). DiskS3 is an experimental feature. #13076 (Pavel Kovalenko).
- Fix readline so it dumps history to file now. #13600 (Amos Bird).
- Create system database with Atomic engine by default (a preparation to enable Atomic database engine by default everywhere). #13680 (tavplubix).

## Performance Improvement

- Slightly optimize very short queries with LowCardinality. #14129 (Anton Popov).
- Enable parallel INSERTs for table engines Null, Memory, Distributed and Buffer when the setting max\_insert\_threads is set. #14120 (alexey-milovidov).

- Fail fast if `max_rows_to_read` limit is exceeded on parts scan. The motivation behind this change is to skip ranges scan for all selected parts if it is clear that `max_rows_to_read` is already exceeded. The change is quite noticeable for queries over big number of parts. [#13677](#) ([Roman Khavronenko](#)).
- Slightly improve performance of aggregation by `UInt8/UInt16` keys. [#13099](#) ([alexey-milovidov](#)).
- Optimize `has()`, `indexOf()` and `countEqual()` functions for `Array(LowCardinality(T))` and constant right arguments. [#12550](#) ([myrrc](#)).
- When performing trivial `INSERT SELECT` queries, automatically set `max_threads` to 1 or `max_insert_threads`, and set `max_block_size` to `min_insert_block_size_rows`. Related to [#5907](#). [#12195](#) ([flynn](#)).

## Experimental Feature

- ClickHouse can work as MySQL replica - it is implemented by `MaterializeMySQL` database engine. Implements [#4006](#). [#10851](#) ([Winter Zhang](#)).
- Add types `Int128`, `Int256`, `UInt256` and related functions for them. Extend Decimals with `Decimal256` (precision up to 76 digits). New types are under the setting `allow_experimental_bigint_types`. It is working extremely slow and bad. The implementation is incomplete. Please don't use this feature. [#13097](#) ([Artem Zuikov](#)).

## Build/Testing/Packaging Improvement

- Added `clickhouse install` script, that is useful if you only have a single binary. [#13528](#) ([alexey-milovidov](#)).
- Allow to run `clickhouse` binary without configuration. [#13515](#) ([alexey-milovidov](#)).
- Enable check for typos in code with `codespell`. [#13513](#) [#13511](#) ([alexey-milovidov](#)).
- Enable Shellcheck in CI as a linter of .sh tests. This closes [#13168](#). [#13530](#) [#13529](#) ([alexey-milovidov](#)).
- Add a CMake option to fail configuration instead of auto-reconfiguration, enabled by default. [#13687](#) ([Konstantin](#)).
- Expose version of embedded tzdata via `TZDATA_VERSION` in `system.build_options`. [#13648](#) ([filimonov](#)).
- Improve generation of `system.time_zones` table during build. Closes [#14209](#). [#14215](#) ([filimonov](#)).
- Build ClickHouse with the most fresh tzdata from package repository. [#13623](#) ([alexey-milovidov](#)).
- Add the ability to write js-style comments in `skip_list.json`. [#14159](#) ([alesapin](#)).
- Ensure that there is no copy-pasted GPL code. [#13514](#) ([alexey-milovidov](#)).
- Switch tests docker images to use test-base parent. [#14167](#) ([Ilya Yatsishin](#)).
- Adding retry logic when bringing up docker-compose cluster; Increasing `COMPOSE_HTTP_TIMEOUT`. [#14112](#) ([vzakaznikov](#)).
- Enabled `system.text_log` in stress test to find more bugs. [#13855](#) ([Nikita Mikhaylov](#)).
- Testflows LDAP module: adding missing certificates and `dhpamparam.pem` for `openldap4`. [#13780](#) ([vzakaznikov](#)).
- ZooKeeper cannot work reliably in unit tests in CI infrastructure. Using unit tests for ZooKeeper interaction with real ZooKeeper is bad idea from the start (unit tests are not supposed to verify complex distributed systems). We already using integration tests for this purpose and they are better suited. [#13745](#) ([alexey-milovidov](#)).
- Added docker image for style check. Added style check that all docker and docker compose files are located in docker directory. [#13724](#) ([Ilya Yatsishin](#)).

- Fix cassandra build on Mac OS. [#13708](#) ([Ilya Yatsishin](#)).
- Fix link error in shared build. [#13700](#) ([Amos Bird](#)).
- Updating LDAP user authentication suite to check that it works with RBAC. [#13656](#) ([vzakaznikov](#)).
- Removed `-DENABLE_CURL_CLIENT` for `contrib/aws`. [#13628](#) ([Vladimir Chebotarev](#)).
- Increasing health-check timeouts for ClickHouse nodes and adding support to dump docker-compose logs if unhealthy containers found. [#13612](#) ([vzakaznikov](#)).
- Make sure [#10977](#) is invalid. [#13539](#) ([Amos Bird](#)).
- Skip PR's from robot-clickhouse. [#13489](#) ([Nikita Mikhaylov](#)).
- Move Dockerfiles from integration tests to `docker/test` directory. `docker_compose` files are available in `runner` docker container. Docker images are built in CI and not in integration tests. [#13448](#) ([Ilya Yatsishin](#)).

## ClickHouse release 20.7

### ClickHouse release v20.7.2.30-stable, 2020-08-31

#### Backward Incompatible Change

- Function `modulo` (operator `%`) with at least one floating point number as argument will calculate remainder of division directly on floating point numbers without converting both arguments to integers. It makes behaviour compatible with most of DBMS. This also applicable for Date and DateTime data types. Added alias `mod`. This closes [#7323](#). [#12585](#) ([alexey-milovidov](#)).
- Deprecate special printing of zero Date/DateTime values as `0000-00-00` and `0000-00-00 00:00:00`. [#12442](#) ([alexey-milovidov](#)).
- The function `groupArrayMoving*` was not working for distributed queries. Its result was calculated within incorrect data type (without promotion to the largest type). The function `groupArrayMovingAvg` was returning integer number that was inconsistent with the `avg` function. This fixes [#12568](#). [#12622](#) ([alexey-milovidov](#)).
- Add sanity check for MergeTree settings. If the settings are incorrect, the server will refuse to start or to create a table, printing detailed explanation to the user. [#13153](#) ([alexey-milovidov](#)).
- Protect from the cases when user may set `background_pool_size` to value lower than `number_of_free_entries_in_pool_to_execute_mutation` or `number_of_free_entries_in_pool_to_lower_max_size_of_merge`. In these cases ALTERs won't work or the maximum size of merge will be too limited. It will throw exception explaining what to do. This closes [#10897](#). [#12728](#) ([alexey-milovidov](#)).
- When upgrading from versions older than 20.5, if rolling update is performed and cluster contains both versions 20.5 or greater and less than 20.5, if ClickHouse nodes with old versions are restarted and old version has been started up in presence of newer versions, it may lead to `Part ... intersects previous part` errors. To prevent this error, first install newer clickhouse-server packages on all cluster nodes and then do restarts (so, when clickhouse-server is restarted, it will start up with the new version).

#### New Feature

- Polygon dictionary type that provides efficient "reverse geocoding" lookups - to find the region by coordinates in a dictionary of many polygons (world map). It is using carefully optimized algorithm with recursive grids to maintain low CPU and memory usage. [#9278](#) ([achulkov2](#)).
- Added support of LDAP authentication for preconfigured users ("Simple Bind" method). [#11234](#) ([Denis Glazachev](#)).

- Introduce setting `alter_partition_verbose_result` which outputs information about touched parts for some types of `ALTER TABLE ... PARTITION ...` queries (currently `ATTACH` and `FREEZE`). Closes #8076. #13017 (alesapin).
- Add `bayesAB` function for bayesian-ab-testing. #12327 (achimbab).
- Added `system.crash_log` table into which stack traces for fatal errors are collected. This table should be empty. #12316 (alexey-milovidov).
- Added http headers `X-ClickHouse-Database` and `X-ClickHouse-Format` which may be used to set default database and output format. #12981 (hcz).
- Add `minMap` and `maxMap` functions support to `SimpleAggregateFunction`. #12662 (Ildus Kurbangaliev).
- Add setting `allow_non_metadata_alters` which restricts to execute `ALTER` queries which modify data on disk. Disabled by default. Closes #11547. #12635 (alesapin).
- A function `formatRow` is added to support turning arbitrary expressions into a string via given format. It's useful for manipulating SQL outputs and is quite versatile combined with the `columns` function. #12574 (Amos Bird).
- Add `FROM_UNIXTIME` function for compatibility with MySQL, related to 12149. #12484 (flynn).
- Allow Nullable types as keys in MergeTree tables if `allow_nullable_key` table setting is enabled. Closes #5319. #12433 (Amos Bird).
- Integration with COS. #12386 (fastio).
- Add `mapAdd` and `mapSubtract` functions for adding/subtracting key-mapped values. #11735 (Ildus Kurbangaliev).

## Bug Fix

- Fix premature `ON CLUSTER` timeouts for queries that must be executed on a single replica. Fixes #6704, #7228, #13361, #11884. #13450 (alesapin).
- Fix crash in mark inclusion search introduced in #12277. #14225 (Amos Bird).
- Fix race condition in external dictionaries with cache layout which can lead server crash. #12566 (alesapin).
- Fix visible data clobbering by progress bar in client in interactive mode. This fixes #12562 and #13369 and #13584 and fixes #12964. #13691 (alexey-milovidov).
- Fixed incorrect sorting order for `LowCardinality` columns when `ORDER BY` multiple columns is used. This fixes #13958. #14223 (Nikita Mikhaylov).
- Removed hardcoded timeout, which wrongly overruled `query_wait_timeout_milliseconds` setting for cache-dictionary. #14105 (Nikita Mikhaylov).
- Fixed wrong mount point in extra info for `Poco::Exception: no space left on device` #14050 (tavplubix).
- Fix wrong query optimization of select queries with `DISTINCT` keyword when subqueries also have `DISTINCT` in case `optimize_duplicate_order_by_and_distinct` setting is enabled. #13925 (Artem Zuikov).
- Fixed potential deadlock when renaming `Distributed` table. #13922 (tavplubix).
- Fix incorrect sorting for `FixedString` columns when `ORDER BY` multiple columns is used. Fixes #13182. #13887 (Nikolai Kochetov).
- Fix potentially lower precision of `topK`/`topKWeighted` aggregations (with non-default parameters). #13817 (Azat Khuzhin).

- Fix reading from MergeTree table with INDEX of type SET fails when compared against NULL. This fixes #13686. #13793 (Amos Bird).
- Fix step overflow in function `range()`. #13790 (Azat Khuzhin).
- Fixed `Directory not empty` error when concurrently executing `DROP DATABASE` and `CREATE TABLE`. #13756 (alexey-milovidov).
- Add range check for `h3KRing` function. This fixes #13633. #13752 (alexey-milovidov).
- Fix race condition between DETACH and background merges. Parts may revive after detach. This is continuation of #8602 that did not fix the issue but introduced a test that started to fail in very rare cases, demonstrating the issue. #13746 (alexey-milovidov).
- Fix logging Settings.Names/Values when `log_queries_min_type` greater than `QUERY_START`. #13737 (Azat Khuzhin).
- Fix incorrect message in `clickhouse-server.init` while checking user and group. #13711 (ylchou).
- Do not optimize `any(arrayJoin())` to `arrayJoin()` under `optimize_move_functions_out_of_any`. #13681 (Azat Khuzhin).
- Fixed possible deadlock in concurrent `ALTER ... REPLACE/MOVE PARTITION ...` queries. #13626 (tavplubix).
- Fixed the behaviour when sometimes cache-dictionary returned default value instead of present value from source. #13624 (Nikita Mikhaylov).
- Fix secondary indices corruption in compact parts (compact parts is an experimental feature). #13538 (Anton Popov).
- Fix wrong code in function `netloc`. This fixes #13335. #13446 (alexey-milovidov).
- Fix error in `parseDateTimeBestEffort` function when unix timestamp was passed as an argument. This fixes #13362. #13441 (alexey-milovidov).
- Fix invalid return type for comparison of tuples with `NULL` elements. Fixes #12461. #13420 (Nikolai Kochetov).
- Fix wrong optimization caused aggregate function `any(x)` is found inside another aggregate function in `queryerror` with `SET optimize_move_functions_out_of_any = 1` and aliases inside `any()`. #13419 (Artem Zuikov).
- Fix possible race in `StorageMemory`. #13416 (Nikolai Kochetov).
- Fix empty output for Arrow and Parquet formats in case if query return zero rows. It was done because empty output is not valid for this formats. #13399 (hcz).
- Fix select queries with constant columns and prefix of primary key in `ORDER BY` clause. #13396 (Anton Popov).
- Fix `PrettyCompactMonoBlock` for clickhouse-local. Fix extremes/totals with `PrettyCompactMonoBlock`. Fixes #7746. #13394 (Azat Khuzhin).
- Fixed deadlock in `system.text_log`. #12452 (alexey-milovidov). It is a part of #12339. This fixes #12325. #13386 (Nikita Mikhaylov).
- Fixed `File(TSVWithNames*)` (header was written multiple times), fixed `clickhouse-local --format CSVWithNames*` (lacks header, broken after #12197), fixed `clickhouse-local --format CSVWithNames*` with zero rows (lacks header). #13343 (Azat Khuzhin).
- Fix segfault when function `groupArrayMovingSum` deserializes empty state. Fixes #13339. #13341 (alesapin).

- Throw error on `arrayJoin()` function in `JOIN ON` section. #13330 (Artem Zuikov).
- Fix crash in `LEFT ASOF JOIN` with `join_use_nulls=1`. #13291 (Artem Zuikov).
- Fix possible error `Totals` having transform was already added to pipeline in case of a query from delayed replica. #13290 (Nikolai Kochetov).
- The server may crash if user passed specifically crafted arguments to the function `h3ToChildren`. This fixes #13275. #13277 (alexey-milovidov).
- Fix potentially low performance and slightly incorrect result for `uniqExact`, `topK`, `sumDistinct` and similar aggregate functions called on `Float` types with `NaN` values. It also triggered assert in debug build. This fixes #12491. #13254 (alexey-milovidov).
- Fix assertion in `KeyCondition` when primary key contains expression with monotonic function and query contains comparison with constant whose type is different. This fixes #12465. #13251 (alexey-milovidov).
- Return passed number for numbers with MSB set in function `roundUpToPowerOfTwoOrZero()`. It prevents potential errors in case of overflow of array sizes. #13234 (Azat Khuzhin).
- Fix function if with nullable `constexpr` as cond that is not literal `NULL`. Fixes #12463. #13226 (alexey-milovidov).
- Fix assert in `arrayElement` function in case of array elements are `Nullable` and array subscript is also `Nullable`. This fixes #12172. #13224 (alexey-milovidov).
- Fix `DateTime64` conversion functions with constant argument. #13205 (Azat Khuzhin).
- Fix parsing row policies from `users.xml` when names of databases or tables contain dots. This fixes #5779, #12527. #13199 (Vitaly Baranov).
- Fix access to `redis` dictionary after connection was dropped once. It may happen with `cache` and `direct` dictionary layouts. #13082 (Anton Popov).
- Fix wrong index analysis with functions. It could lead to some data parts being skipped when reading from `MergeTree` tables. Fixes #13060. Fixes #12406. #13081 (Anton Popov).
- Fix error `Cannot convert column because it is constant but values of constants are different in source and result` for remote queries which use deterministic functions in scope of query, but not deterministic between queries, like `now()`, `now64()`, `randConstant()`. Fixes #11327. #13075 (Nikolai Kochetov).
- Fix crash which was possible for queries with `ORDER BY tuple` and small `LIMIT`. Fixes #12623. #13009 (Nikolai Kochetov).
- Fix `Block structure mismatch` error for queries with `UNION` and `JOIN`. Fixes #12602. #12989 (Nikolai Kochetov).
- Corrected `merge_with_ttl_timeout` logic which did not work well when expiration affected more than one partition over one time interval. (Authored by @excitoon). #12982 (Alexander Kazakov).
- Fix columns duplication for range hashed dictionary created from DDL query. This fixes #10605. #12857 (alesapin).
- Fix unnecessary limiting for the number of threads for selects from local replica. #12840 (Nikolai Kochetov).

- Fix rare bug when `ALTER DELETE` and `ALTER MODIFY COLUMN` queries executed simultaneously as a single mutation. Bug leads to an incorrect amount of rows in `count.txt` and as a consequence incorrect data in part. Also, fix a small bug with simultaneous `ALTER RENAME COLUMN` and `ALTER ADD COLUMN`. [#12760](#) ([alesapin](#)).
- Wrong credentials being used when using `clickhouse` dictionary source to query remote tables. [#12755](#) ([sundylis](#)).
- Fix `CAST(Nullable(String), Enum())`. [#12745](#) ([Azat Khuzhin](#)).
- Fix performance with large tuples, which are interpreted as functions in `IN` section. The case when user writes `WHERE x IN tuple(1, 2, ...)` instead of `WHERE x IN (1, 2, ...)` for some obscure reason. [#12700](#) ([Anton Popov](#)).
- Fix memory tracking for `input_format_parallel_parsing` (by attaching thread to group). [#12672](#) ([Azat Khuzhin](#)).
- Fix wrong optimization `optimize_move_functions_out_of_any=1` in case of `any(func(<lambda>))`. [#12664](#) ([Artem Zuikov](#)).
- Fixed [#10572](#) fix bloom filter index with const expression. [#12659](#) ([Winter Zhang](#)).
- Fix `SIGSEGV` in `StorageKafka` when broker is unavailable (and not only). [#12658](#) ([Azat Khuzhin](#)).
- Add support for function `if` with `Array(UUID)` arguments. This fixes [#11066](#). [#12648](#) ([alexey-milovidov](#)).
- `CREATE USER IF NOT EXISTS` now does not throw exception if the user exists. This fixes [#12507](#). [#12646](#) ([Vitaly Baranov](#)).
- Exception `There is no supertype...` can be thrown during `ALTER ... UPDATE` in unexpected cases (e.g. when subtracting from `UInt64` column). This fixes [#7306](#). This fixes [#4165](#). [#12633](#) ([alexey-milovidov](#)).
- Fix possible `Pipeline` stuck error for queries with external sorting. Fixes [#12617](#). [#12618](#) ([Nikolai Kochetov](#)).
- Fix error `Output of TreeExecutor is not sorted` for `OPTIMIZE DEDUPLICATE`. Fixes [#11572](#). [#12613](#) ([Nikolai Kochetov](#)).
- Fix the issue when alias on result of function `any` can be lost during query optimization. [#12593](#) ([Anton Popov](#)).
- Remove data for Distributed tables (blocks from async `INSERTs`) on `DROP TABLE`. [#12556](#) ([Azat Khuzhin](#)).
- Now ClickHouse will recalculate checksums for parts when file `checksums.txt` is absent. Broken since [#9827](#). [#12545](#) ([alesapin](#)).
- Fix bug which lead to broken old parts after `ALTER DELETE` query when `enable_mixed_granularity_parts=1`. Fixes [#12536](#). [#12543](#) ([alesapin](#)).
- Fixing race condition in live view tables which could cause data duplication. `LIVE VIEW` is an experimental feature. [#12519](#) ([vzakaznikov](#)).
- Fix backwards compatibility in binary format of `AggregateFunction(avg, ...)` values. This fixes [#12342](#). [#12486](#) ([alexey-milovidov](#)).
- Fix crash in `JOIN` with dictionary when we are joining over expression of dictionary key: `t JOIN dict ON expr(dict.id) = t.id`. Disable dictionary join optimisation for this case. [#12458](#) ([Artem Zuikov](#)).
- Fix overflow when very large `LIMIT` or `OFFSET` is specified. This fixes [#10470](#). This fixes [#11372](#). [#12427](#) ([alexey-milovidov](#)).

- kafka: fix SIGSEGV if there is a message with error in the middle of the batch. #12302 (Azat Khuzhin).

## Improvement

- Keep smaller amount of logs in ZooKeeper. Avoid excessive growing of ZooKeeper nodes in case of offline replicas when having many servers/tables/inserts. #13100 (alexey-milovidov).
- Now exceptions forwarded to the client if an error happened during ALTER or mutation. Closes #11329. #12666 (alesapin).
- Add `QueryTimeMicroseconds`, `SelectQueryTimeMicroseconds` and `InsertQueryTimeMicroseconds` to `system.events`, along with `system.metrics`, `processes`, `query_log`, etc. #13028 (ianton-ru).
- Added `SelectedRows` and `SelectedBytes` to `system.events`, along with `system.metrics`, `processes`, `query_log`, etc. #12638 (ianton-ru).
- Added `current_database` information to `system.query_log`. #12652 (Amos Bird).
- Allow `TabSeparatedRaw` as input format. #12009 (hczi).
- Now `joinGet` supports multi-key lookup. #12418 (Amos Bird).
- Allow `*Map` aggregate functions to work on Arrays with NULLs. Fixes #13157. #13225 (alexey-milovidov).
- Avoid overflow in parsing of DateTime values that will lead to negative unix timestamp in their timezone (for example, 1970-01-01 00:00:00 in Moscow). Saturate to zero instead. This fixes #3470. This fixes #4172. #12443 (alexey-milovidov).
- AvroConfluent: Skip Kafka tombstone records - Support skipping broken records #13203 (Andrew Onyshchuk).
- Fix wrong error for long queries. It was possible to get syntax error other than Max query size exceeded for correct query. #13928 (Nikolai Kochetov).
- Fix data race in `Igamma` function. This race was caught only in `tsan`, no side effects really happened. #13842 (Nikolai Kochetov).
- Fix a 'Week'-interval formatting for ATTACH/ALTER/CREATE QUOTA-statements. #13417 (vladimir-golovchenko).
- Now broken parts are also reported when encountered in compact part processing. Compact parts is an experimental feature. #13282 (Amos Bird).
- Fix assert in `geohashesInBox`. This fixes #12554. #13229 (alexey-milovidov).
- Fix assert in `parseDateTimeBestEffort`. This fixes #12649. #13227 (alexey-milovidov).
- Minor optimization in Processors/PipelineExecutor: breaking out of a loop because it makes sense to do so. #13058 (Mark Papadakis).
- Support TRUNCATE table without TABLE keyword. #12653 (Winter Zhang).
- Fix explain query format overwrite by default. This fixes #12541. #12541 (BohuTANG).
- Allow to set JOIN kind and type in more standad way: `LEFT SEMI JOIN` instead of `SEMI LEFT JOIN`. For now both are correct. #12520 (Artem Zuikov).
- Changes default value for `multiple_joins_rewriter_version` to 2. It enables new multiple joins rewriter that knows about column names. #12469 (Artem Zuikov).
- Add several metrics for requests to S3 storages. #12464 (ianton-ru).

- Use correct default secure port for clickhouse-benchmark with `--secure` argument. This fixes #11044. #12440 (alexey-milovidov).
- Rollback insertion errors in `Log`, `TinyLog`, `StripeLog` engines. In previous versions insertion error lead to inconsistent table state (this works as documented and it is normal for these table engines). This fixes #12402. #12426 (alexey-milovidov).
- Implement `RENAME DATABASE` and `RENAME DICTIONARY` for Atomic database engine - Add implicit `{uuid}` macro, which can be used in ZooKeeper path for ReplicatedMergeTree. It works with `CREATE ... ON CLUSTER ...` queries. Set `show_table_uuid_in_table_create_query_if_not_nil` to true to use it. - Make `ReplicatedMergeTree` engine arguments optional, `/clickhouse/tables/{uuid}/{shard}/` and `{replica}` are used by default. Closes #12135. - Minor fixes. - These changes break backward compatibility of Atomic database engine. Previously created Atomic databases must be manually converted to new format. Atomic database is an experimental feature. #12343 (tavplubix).
- Separated `AWSAuthV4Signer` into different logger, removed excessive `AWSClient`: `AWSClient` from log messages. #12320 (Vladimir Chebotarev).
- Better exception message in disk access storage. #12625 (alesapin).
- Better exception for function `in` with invalid number of arguments. #12529 (Anton Popov).
- Fix error message about adaptive granularity. #12624 (alesapin).
- Fix SETTINGS parse after `FORMAT`. #12480 (Azat Khuzhin).
- If MergeTree table does not contain `ORDER BY` or `PARTITION BY`, it was possible to request `ALTER` to `CLEAR` all the columns and `ALTER` will stuck. Fixed #7941. #12382 (alexey-milovidov).
- Avoid re-loading completion from the history file after each query (to avoid history overlaps with other client sessions). #13086 (Azat Khuzhin).

## Performance Improvement

- Lower memory usage for some operations up to 2 times. #12424 (alexey-milovidov).
- Optimize PK lookup for queries that match exact PK range. #12277 (Ivan Babrou).
- Slightly optimize very short queries with `LowCardinality`. #14129 (Anton Popov).
- Slightly improve performance of aggregation by `UInt8/UInt16` keys. #13091 and #13055 (alexey-milovidov).
- Push down `LIMIT` step for query plan (inside subqueries). #13016 (Nikolai Kochetov).
- Parallel primary key lookup and skipping index stages on parts, as described in #11564. #12589 (Ivan Babrou).
- Converting String-type arguments of function "if" and "transform" into enum if `set optimize_if_transform_strings_to_enum = 1`. #12515 (Artem Zuikov).
- Replaces monotonic functions with its argument in `ORDER BY` if `set optimize_monotonous_functions_in_order_by=1`. #12467 (Artem Zuikov).
- Add order by optimization that rewrites `ORDER BY x, f(x)` with `ORDER by x` if `set optimize_redundant_functions_in_order_by = 1`. #12404 (Artem Zuikov).
- Allow pushdown predicate when subquery contains `WITH` clause. This fixes #12293 #12663 (Winter Zhang).

- Improve performance of reading from compact parts. Compact parts is an experimental feature. #12492 (Anton Popov).
- Attempt to implement streaming optimization in `DiskS3`. DiskS3 is an experimental feature. #12434 (Vladimir Chebotarev).

## Build/Testing/Packaging Improvement

- Use `shellcheck` for sh tests linting. #13200 #13207 (alexey-milovidov).
- Add script which set labels for pull requests in GitHub hook. #13183 (alesapin).
- Remove some of recursive submodules. See #13378. #13379 (alexey-milovidov).
- Ensure that all the submodules are from proper URLs. Continuation of #13379. This fixes #13378. #13397 (alexey-milovidov).
- Added support for user-declared settings, which can be accessed from inside queries. This is needed when ClickHouse engine is used as a component of another system. #13013 (Vitaly Baranov).
- Added testing for RBAC functionality of `INSERT` privilege in TestFlows. Expanded tables on which `SELECT` is being tested. Added Requirements to match new table engine tests. #13340 (MyroTk).
- Fix timeout error during server restart in the stress test. #13321 (alesapin).
- Now fast test will wait server with retries. #13284 (alesapin).
- Function `materialize()` (the function for ClickHouse testing) will work for `NULL` as expected - by transforming it to non-constant column. #13212 (alexey-milovidov).
- Fix libunwind build in AArch64. This fixes #13204. #13208 (alexey-milovidov).
- Even more retries in zkutil gtest to prevent test flakiness. #13165 (alexey-milovidov).
- Small fixes to the RBAC TestFlows. #13152 (vzakaznikov).
- Fixing `00960_live_view_watch_events_live.py` test. #13108 (vzakaznikov).
- Improve cache purge in documentation deploy script. #13107 (alesapin).
- Rewrote some orphan tests to gtest. Removed useless includes from tests. #13073 (Nikita Mikhaylov).
- Added tests for RBAC functionality of `SELECT` privilege in TestFlows. #13061 (Ritaank Tiwari).
- Rerun some tests in fast test check. #12992 (alesapin).
- Fix MSan error in "rdkafka" library. This closes #12990. Updated `rdkafka` to version 1.5 (master). #12991 (alexey-milovidov).
- Fix UBSan report in `base64` if tests were run on server with AVX-512. This fixes #12318. Author: @qoega. #12441 (alexey-milovidov).
- Fix UBSan report in `HDFS` library. This closes #12330. #12453 (alexey-milovidov).
- Check an ability that we able to restore the backup from an old version to the new version. This closes #8979. #12959 (alesapin).
- Do not build `helper_container` image inside integrational tests. Build docker container in CI and use pre-built `helper_container` in integration tests. #12953 (Ilya Yatsishin).
- Add a test for `ALTER TABLE CLEAR COLUMN` query for primary key columns. #12951 (alesapin).
- Increased timeouts in testflows tests. #12949 (vzakaznikov).

- Fix build of test under Mac OS X. This closes #12767. #12772 (alexey-milovidov).
- Connector-ODBC updated to mysql-connector-odbc-8.0.21. #12739 (Ilya Yatsishin).
- Adding RBAC syntax tests in TestFlows. #12642 (vzakaznikov).
- Improve performance of TestKeeper. This will speedup tests with heavy usage of Replicated tables. #12505 (alexey-milovidov).
- Now we check that server is able to start after stress tests run. This fixes #12473. #12496 (alesapin).
- Update fmtlib to master (7.0.1). #12446 (alexey-milovidov).
- Add docker image for fast tests. #12294 (alesapin).
- Rework configuration paths for integration tests. #12285 (Ilya Yatsishin).
- Add compiler option to control that stack frames are not too large. This will help to run the code in fibers with small stack size. #11524 (alexey-milovidov).
- Update gitignore-files. #13447 (vladimir-golovchenko).

## ClickHouse release 20.6

### ClickHouse release v20.6.3.28-stable

#### Backward Incompatible Change

- When upgrading from versions older than 20.5, if rolling update is performed and cluster contains both versions 20.5 or greater and less than 20.5, if ClickHouse nodes with old versions are restarted and old version has been started up in presence of newer versions, it may lead to `Part ... intersects previous part` errors. To prevent this error, first install newer clickhouse-server packages on all cluster nodes and then do restarts (so, when clickhouse-server is restarted, it will start up with the new version).

#### New Feature

- Added an initial implementation of EXPLAIN query. Syntax: `EXPLAIN SELECT ....` This fixes #1118. #11873 (Nikolai Kochetov).
- Added storage RabbitMQ. #11069 (Kseniia Sumarokova).
- Implemented PostgreSQL-like `ILIKE` operator for #11710. #12125 (Mike).
- Supported RIGHT and FULL JOIN with `SET join_algorithm = 'partial_merge'`. Only ALL strictness is allowed (ANY, SEMI, ANTI, ASOF are not). #12118 (Artem Zuikov).
- Added a function `initializeAggregation` to initialize an aggregation based on a single value. #12109 (Guillaume Tassery).
- Supported `ALTER TABLE ... [ADD|MODIFY] COLUMN ... FIRST`#4006. #12073 (Winter Zhang).
- Added function `parseDateTimeBestEffortUS`. #12028 (flynn).
- Support format ORC for output (was supported only for input). #11662 (Kruglov Pavel).

#### Bug Fix

- Fixed aggregate function `any(x)` is found inside another aggregate function in `queryerror` with `SET optimize_move_functions_out_of_any = 1` and aliases inside `any()`. #13419 (Artem Zuikov).
- Fixed PrettyCompactMonoBlock for clickhouse-local. Fixed extremes/totals with PrettyCompactMonoBlock. This fixes #7746. #13394 (Azat Khuzhin).

- Fixed possible error `Totals` having transform was already added to pipeline in case of a query from delayed replica. #13290 (Nikolai Kochetov).
- The server may crash if user passed specifically crafted arguments to the function `h3ToChildren`. This fixes #13275. #13277 (alexey-milovidov).
- Fixed potentially low performance and slightly incorrect result for `uniqExact`, `topK`, `sumDistinct` and similar aggregate functions called on `Float` types with `Nan` values. It also triggered assert in debug build. This fixes #12491. #13254 (alexey-milovidov).
- Fixed function if with nullable `constexpr` as cond that is not literal `NULL`. Fixes #12463. #13226 (alexey-milovidov).
- Fixed assert in `arrayElement` function in case of array elements are `Nullable` and array subscript is also `Nullable`. This fixes #12172. #13224 (alexey-milovidov).
- Fixed `DateTime64` conversion functions with constant argument. #13205 (Azat Khuzhin).
- Fixed wrong index analysis with functions. It could lead to pruning wrong parts, while reading from `MergeTree` tables. Fixes #13060. Fixes #12406. #13081 (Anton Popov).
- Fixed error Cannot convert column because it is constant but values of constants are different in source and result for remote queries which use deterministic functions in scope of query, but not deterministic between queries, like `now()`, `now64()`, `randConstant()`. Fixes #11327. #13075 (Nikolai Kochetov).
- Fixed unnecessary limiting for the number of threads for selects from local replica. #12840 (Nikolai Kochetov).
- Fixed rare bug when `ALTER DELETE` and `ALTER MODIFY COLUMN` queries executed simultaneously as a single mutation. Bug leads to an incorrect amount of rows in `count.txt` and as a consequence incorrect data in part. Also, fix a small bug with simultaneous `ALTER RENAME COLUMN` and `ALTER ADD COLUMN`. #12760 (alesapin).
- Fixed `CAST(Nullable(String), Enum())`. #12745 (Azat Khuzhin).
- Fixed a performance with large tuples, which are interpreted as functions in `IN` section. The case when user write `WHERE x IN tuple(1, 2, ...)` instead of `WHERE x IN (1, 2, ...)` for some obscure reason. #12700 (Anton Popov).
- Fixed memory tracking for `input_format_parallel_parsing` (by attaching thread to group). #12672 (Azat Khuzhin).
- Fixed bloom filter index with const expression. This fixes #10572. #12659 (Winter Zhang).
- Fixed `SIGSEGV` in `StorageKafka` when broker is unavailable (and not only). #12658 (Azat Khuzhin).
- Added support for function if with `Array(UUID)` arguments. This fixes #11066. #12648 (alexey-milovidov).
- `CREATE USER IF NOT EXISTS` now does not throw exception if the user exists. This fixes #12507. #12646 (Vitaly Baranov).
- Better exception message in disk access storage. #12625 (alesapin).
- The function `groupArrayMoving*` was not working for distributed queries. Its result was calculated within incorrect data type (without promotion to the largest type). The function `groupArrayMovingAvg` was returning integer number that was inconsistent with the `avg` function. This fixes #12568. #12622 (alexey-milovidov).
- Fixed lack of aliases with function `any`. #12593 (Anton Popov).

- Fixed race condition in external dictionaries with cache layout which can lead server crash. #12566 (alesapin).
- Remove data for Distributed tables (blocks from async INSERTs) on DROP TABLE. #12556 (Azat Khuzhin).
- Fixed bug which lead to broken old parts after `ALTER DELETE` query when `enable_mixed_granularity_parts=1`. Fixes #12536. #12543 (alesapin).
- Better exception for function `in` with invalid number of arguments. #12529 (Anton Popov).
- Fixing race condition in live view tables which could cause data duplication. #12519 (vzakaznikov).
- Fixed performance issue, while reading from compact parts. #12492 (Anton Popov).
- Fixed backwards compatibility in binary format of `AggregateFunction(avg, ...)` values. This fixes #12342. #12486 (alexey-milovidov).
- Fixed SETTINGS parse after FORMAT. #12480 (Azat Khuzhin).
- Fixed the deadlock if `text_log` is enabled. #12452 (alexey-milovidov).
- Fixed overflow when very large `LIMIT` or `OFFSET` is specified. This fixes #10470. This fixes #11372. #12427 (alexey-milovidov).
- Fixed possible segfault if `StorageMerge`. This fixes #12054. #12401 (tavplubix).
- Reverted change introduced in #11079 to resolve #12098. #12397 (Mike).
- Additional check for arguments of bloom filter index. This fixes #11408. #12388 (alexey-milovidov).
- Avoid exception when negative or floating point constant is used in WHERE condition for indexed tables. This fixes #11905. #12384 (alexey-milovidov).
- Allowed to `CLEAR` column even if there are depending `DEFAULT` expressions. This fixes #12333. #12378 (alexey-milovidov).
- Fix `TOTALS/ROLLUP/CUBE` for aggregate functions with `-State` and `Nullable` arguments. This fixes #12163. #12376 (alexey-milovidov).
- Fixed error message and exit codes for `ALTER RENAME COLUMN` queries, when `RENAME` is not allowed. Fixes #12301 and #12303. #12335 (alesapin).
- Fixed very rare race condition in `ReplicatedMergeTreeQueue`. #12315 (alexey-milovidov).
- When using codec `Delta` or `DoubleDelta` with non fixed width types, exception with code `LOGICAL_ERROR` was returned instead of exception with code `BAD_ARGUMENTS` (we ensure that exceptions with code logical error never happen). This fixes #12110. #12308 (alexey-milovidov).
- Fixed order of columns in `WITH FILL` modifier. Previously order of columns of `ORDER BY` statement wasn't respected. #12306 (Anton Popov).
- Avoid "bad cast" exception when there is an expression that filters data by virtual columns (like `_table` in `Merge` tables) or by "index" columns in system tables such as filtering by database name when querying from `system.tables`, and this expression returns `Nullable` type. This fixes #12166. #12305 (alexey-milovidov).
- Fixed `TTL` after renaming column, on which depends `TTL` expression. #12304 (Anton Popov).
- Fixed SIGSEGV if there is an message with error in the middle of the batch in Kafka Engine. #12302 (Azat Khuzhin).

- Fixed the situation when some threads might randomly hang for a few seconds during DNS cache updating. #12296 (tavplubix).
- Fixed typo in setting name. #12292 (alexey-milovidov).
- Show error after TrieDictionary failed to load. #12290 (Vitaly Baranov).
- The function `arrayFill` worked incorrectly for empty arrays that may lead to crash. This fixes #12263. #12279 (alexey-milovidov).
- Implement conversions to the common type for `LowCardinality` types. This allows to execute UNION ALL of tables with columns of `LowCardinality` and other columns. This fixes #8212. This fixes #4342. #12275 (alexey-milovidov).
- Fixed the behaviour on reaching redirect limit in request to S3 storage. #12256 (ianton-ru).
- Fixed the behaviour when during multiple sequential inserts in `StorageFile` header for some special types was written more than once. This fixed #6155. #12197 (Nikita Mikhaylov).
- Fixed logical functions for `UInt8` values when they are not equal to 0 or 1. #12196 (Alexander Kazakov).
- Cap `max_memory_usage*` limits to the process resident memory. #12182 (Azat Khuzhin).
- Fix dictGet arguments check during `GROUP BY` injective functions elimination. #12179 (Azat Khuzhin).
- Fixed the behaviour when `SummingMergeTree` engine sums up columns from partition key. Added an exception in case of explicit definition of columns to sum which intersects with partition key columns. This fixes #7867. #12173 (Nikita Mikhaylov).
- Don't split the dictionary source's table name into schema and table name itself if ODBC connection does not support schema. #12165 (Vitaly Baranov).
- Fixed wrong logic in `ALTER DELETE` that leads to deleting of records when condition evaluates to NULL. This fixes #9088. This closes #12106. #12153 (alexey-milovidov).
- Fixed transform of query to send to external DBMS (e.g. MySQL, ODBC) in presence of aliases. This fixes #12032. #12151 (alexey-milovidov).
- Fixed bad code in redundant ORDER BY optimization. The bug was introduced in #10067. #12148 (alexey-milovidov).
- Fixed potential overflow in integer division. This fixes #12119. #12140 (alexey-milovidov).
- Fixed potential infinite loop in `greatCircleDistance`, `geoDistance`. This fixes #12117. #12137 (alexey-milovidov).
- Normalize "pid" file handling. In previous versions the server may refuse to start if it was killed without proper shutdown and if there is another process that has the same pid as previously runned server. Also pid file may be removed in unsuccessful server startup even if there is another server running. This fixes #3501. #12133 (alexey-milovidov).
- Fixed bug which leads to incorrect table metadata in ZooKeepeer for ReplicatedVersionedCollapsingMergeTree tables. Fixes #12093. #12121 (alesapin).
- Avoid "There is no query" exception for materialized views with joins or with subqueries attached to system logs (`system.query_log`, `metric_log`, etc) or to `engine=Buffer` underlying table. #12120 (filimonov).
- Fixed handling dependency of table with `ENGINE=Dictionary` on dictionary. This fixes #10994. This fixes #10397. #12116 (Vitaly Baranov).

- Format Parquet now properly works with `LowCardinality` and `LowCardinality(Nullable)` types. Fixes #12086, #8406. #12108 (Nikolai Kochetov).
- Fixed performance for selects with `UNION` caused by wrong limit for the total number of threads. Fixes #12030. #12103 (Nikolai Kochetov).
- Fixed segfault with `-StateResample` combinators. #12092 (Anton Popov).
- Fixed empty `result_rows` and `result_bytes` metrics in `system.quey_log` for selects. Fixes #11595. #12089 (Nikolai Kochetov).
- Fixed unnecessary limiting the number of threads for selects from `VIEW`. Fixes #11937. #12085 (Nikolai Kochetov).
- Fixed SIGSEGV in StorageKafka on DROP TABLE. #12075 (Azat Khuzhin).
- Fixed possible crash while using wrong type for `PREWHERE`. Fixes #12053, #12060. #12060 (Nikolai Kochetov).
- Fixed error `Cannot capture column` for higher-order functions with `Tuple(LowCardinality)` argument. Fixes #9766. #12055 (Nikolai Kochetov).
- Fixed constraints check if constraint is a constant expression. This fixes #11360. #12042 (alexey-milovidov).
- Fixed wrong result and potential crash when invoking function `if` with arguments of type `FixedString` with different sizes. This fixes #11362. #12021 (alexey-milovidov).

## Improvement

- Allowed to set `JOIN` kind and type in more standard way: `LEFT SEMI JOIN` instead of `SEMI LEFT JOIN`. For now both are correct. #12520 (Artem Zuikov).
- `lifetime_rows/lifetime_bytes` for Buffer engine. #12421 (Azat Khuzhin).
- Write the detail exception message to the client instead of 'MySQL server has gone away'. #12383 (BohuTANG).
- Allows to change a charset which is used for printing grids borders. Available charsets are following: UTF-8, ASCII. Setting `output_format_pretty_grid_charset` enables this feature. #12372 (Sabyanin Maxim).
- Supported MySQL 'SELECT DATABASE()' #9336 2. Add MySQL replacement query integration test. #12314 (BohuTANG).
- Added `KILL QUERY [connection_id]` for the MySQL client/driver to cancel the long query, issue #12038. #12152 (BohuTANG).
- Added support for `%g` (two digit ISO year) and `%G` (four digit ISO year) substitutions in `formatDateTime` function. #12136 (vivarum).
- Added 'type' column in `system.disks`. #12115 (ianton-ru).
- Improved `REVOKE` command: now it requires grant/admin option for only access which will be revoked. For example, to execute `REVOKE ALL ON *.* FROM user1` now it does not require to have full access rights granted with grant option. Added command `REVOKE ALL FROM user1` - it revokes all granted roles from `user1`. #12083 (Vitaly Baranov).
- Added replica priority for `load_balancing` (for manual prioritization of the load balancing). #11995 (Azat Khuzhin).

- Switched paths in S3 metadata to relative which allows to handle S3 blobs more easily. #11892 ([Vladimir Chebotarev](#)).

## Performance Improvement

- Improved performance of 'ORDER BY' and 'GROUP BY' by prefix of sorting key (enabled with `optimize_aggregation_in_order` setting, disabled by default). #11696 ([Anton Popov](#)).
- Removed injective functions inside `uniq*()` if set `optimize_injective_functions_inside_uniq=1`. #12337 ([Ruslan Kamalov](#)).
- Index not used for IN operator with literals, performance regression introduced around v19.3. This fixes #10574. #12062 ([nvartolomei](#)).
- Implemented single part uploads for DiskS3 (experimental feature). #12026 ([Vladimir Chebotarev](#)).

## Experimental Feature

- Added new in-memory format of parts in MergeTree-family tables, which stores data in memory. Parts are written on disk at first merge. Part will be created in in-memory format if its size in rows or bytes is below thresholds `min_rows_for_compact_part` and `min_bytes_for_compact_part`. Also optional support of Write-Ahead-Log is available, which is enabled by default and is controlled by setting `in_memory_parts_enable_wal`. #10697 ([Anton Popov](#)).

## Build/Testing/Packaging Improvement

- Implement AST-based query fuzzing mode for clickhouse-client. See [this label](#) for the list of issues we recently found by fuzzing. Most of them were found by this tool, and a couple by SQLancer and `00746_sql_fuzzy.pl`. #12111 ([Alexander Kuzmenkov](#)).
- Add new type of tests based on Testflows framework. #12090 ([vzakaznikov](#)).
- Added S3 HTTPS integration test. #12412 ([Pavel Kovalenko](#)).
- Log sanitizer trap messages from separate thread. This will prevent possible deadlock under thread sanitizer. #12313 ([alexey-milovidov](#)).
- Now functional and stress tests will be able to run with old version of `clickhouse-test` script. #12287 ([alesapin](#)).
- Remove strange file creation during build in `orc`. #12258 ([Nikita Mikhaylov](#)).
- Place common docker compose files to integration docker container. #12168 ([Ilya Yatsishin](#)).
- Fix warnings from CodeQL. `CodeQL` is another static analyzer that we will use along with `clang-tidy` and PVS-Studio that we use already. #12138 ([alexey-milovidov](#)).
- Minor CMake fixes for UNBUNDLED build. #12131 ([Matwey V. Kornilov](#)).
- Added a showcase of the minimal Docker image without using any Linux distribution. #12126 ([alexey-milovidov](#)).
- Perform an upgrade of system packages in the `clickhouse-server` docker image. #12124 ([Ivan Blinkov](#)).
- Add `UNBUNDLED` flag to `system.build_options` table. Move skip lists for `clickhouse-test` to `clickhouse` repo. #12107 ([alesapin](#)).
- Regular check by [Anchore Container Analysis](#) security analysis tool that looks for [CVE](#) in `clickhouse-server` Docker image. Also confirms that `Dockerfile` is buildable. Runs daily on master and on pull-requests to `Dockerfile`. #12102 ([Ivan Blinkov](#)).
- Daily check by [GitHub CodeQL](#) security analysis tool that looks for [CWE](#). #12101 ([Ivan Blinkov](#)).

- Install `ca-certificates` before the first `apt-get update` in Dockerfile. #12095 (Ivan Blinkov).

## ClickHouse release 20.5

### ClickHouse release v20.5.4.40-stable 2020-08-10

#### Bug Fix

- Fixed wrong index analysis with functions. It could lead to pruning wrong parts, while reading from `MergeTree` tables. Fixes #13060. Fixes #12406. #13081 (Anton Popov).
- Fixed unnecessary limiting for the number of threads for selects from local replica. #12840 (Nikolai Kochetov).
- Fixed performance with large tuples, which are interpreted as functions in `IN` section. The case when user write `WHERE x IN tuple(1, 2, ...)` instead of `WHERE x IN (1, 2, ...)` for some obscure reason. #12700 (Anton Popov).
- Fixed memory tracking for `input_format_parallel_parsing` (by attaching thread to group). #12672 (Azat Khuzhin).
- Fixed bloom filter index with const expression. This fixes #10572. #12659 (Winter Zhang).
- Fixed `SIGSEGV` in `StorageKafka` when broker is unavailable (and not only). #12658 (Azat Khuzhin).
- Added support for function `if` with `Array(UUID)` arguments. This fixes #11066. #12648 (alexey-milovidov).
- Fixed lack of aliases with function `any`. #12593 (Anton Popov).
- Fixed race condition in external dictionaries with cache layout which can lead server crash. #12566 (alesapin).
- Remove data for Distributed tables (blocks from async INSERTs) on `DROP TABLE`. #12556 (Azat Khuzhin).
- Fixed bug which lead to broken old parts after `ALTER DELETE` query when `enable_mixed_granularity_parts=1`. Fixes #12536. #12543 (alesapin).
- Better exception for function `in` with invalid number of arguments. #12529 (Anton Popov).
- Fixed race condition in live view tables which could cause data duplication. #12519 (vzakaznikov).
- Fixed performance issue, while reading from compact parts. #12492 (Anton Popov).
- Fixed backwards compatibility in binary format of `AggregateFunction(avg, ...)` values. This fixes #12342. #12486 (alexey-milovidov).
- Fixed the deadlock if `text_log` is enabled. #12452 (alexey-milovidov).
- Fixed overflow when very large `LIMIT` or `OFFSET` is specified. This fixes #10470. This fixes #11372. #12427 (alexey-milovidov).
- Fixed possible segfault if `StorageMerge`. Closes #12054. #12401 (tavplubix).
- Reverts change introduced in #11079 to resolve #12098. #12397 (Mike).
- Avoid exception when negative or floating point constant is used in `WHERE` condition for indexed tables. This fixes #11905. #12384 (alexey-milovidov).
- Allow to `CLEAR` column even if there are depending `DEFAULT` expressions. This fixes #12333. #12378 (alexey-milovidov).

- Fixed TOTALS/ROLLUP/CUBE for aggregate functions with `-State` and `Nullable` arguments. This fixes #12163. #12376 (alexey-milovidov).
- Fixed SIGSEGV if there is an message with error in the middle of the batch in Kafka Engine. #12302 (Azat Khuzhin).
- Fixed the behaviour when `SummingMergeTree` engine sums up columns from partition key. Added an exception in case of explicit definition of columns to sum which intersects with partition key columns. This fixes #7867. #12173 (Nikita Mikhaylov).
- Fixed transform of query to send to external DBMS (e.g. MySQL, ODBC) in presence of aliases. This fixes #12032. #12151 (alexey-milovidov).
- Fixed bug which leads to incorrect table metadata in ZooKeepeer for ReplicatedVersionedCollapsingMergeTree tables. Fixes #12093. #12121 (alesapin).
- Fixed unnecessary limiting the number of threads for selects from `VIEW`. Fixes #11937. #12085 (Nikolai Kochetov).
- Fixed crash in JOIN with `LowCardinality` type with `join_algorithm=partial_merge`. #12035 (Artem Zuikov).
- Fixed wrong result for `if()` with `NULLs` in condition. #11807 (Artem Zuikov).

## Performance Improvement

- Index not used for `IN` operator with literals, performance regression introduced around v19.3. This fixes #10574. #12062 (nvartolomei).

## Build/Testing/Packaging Improvement

- Install `ca-certificates` before the first `apt-get update` in Dockerfile. #12095 (Ivan Blinkov).

# ClickHouse release v20.5.2.7-stable 2020-07-02

## Backward Incompatible Change

- Return non-`Nullable` result from `COUNT(DISTINCT)`, and `uniq` aggregate functions family. If all passed values are `NULL`, return zero instead. This improves SQL compatibility. #11661 (alexey-milovidov).
- Added a check for the case when user-level setting is specified in a wrong place. User-level settings should be specified in `users.xml` inside `<profile>` section for specific user profile (or in `<default>` for default settings). The server won't start with exception message in log. This fixes #9051. If you want to skip the check, you can either move settings to the appropriate place or add `<skip_check_for_incorrect_settings>1</skip_check_for_incorrect_settings>` to `config.xml`. #11449 (alexey-milovidov).
- The setting `input_format_with_names_use_header` is enabled by default. It will affect parsing of input formats `-WithNames` and `-WithNamesAndTypes`. #10937 (alexey-milovidov).
- Remove `experimental_use_processors` setting. It is enabled by default. #10924 (Nikolai Kochetov).
- 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. This change is backward compatible but we list it here in changelog in case you will wonder about these messages. #10663 (alexey-milovidov).
- Added a check for meaningless codecs and a setting `allow_suspicious_codecs` to control this check. This closes #4966. #10645 (alexey-milovidov).
- Several Kafka setting changes their defaults. See #11388.

- When upgrading from versions older than 20.5, if rolling update is performed and cluster contains both versions 20.5 or greater and less than 20.5, if ClickHouse nodes with old versions are restarted and old version has been started up in presence of newer versions, it may lead to `Part ... intersects previous part` errors. To prevent this error, first install newer clickhouse-server packages on all cluster nodes and then do restarts (so, when clickhouse-server is restarted, it will start up with the new version).

## New Feature

- `TTL DELETE WHERE` and `TTL GROUP BY` for automatic data coarsening and rollup in tables. [#10537](#) ([expl0si0nn](#)).
- Implementation of PostgreSQL wire protocol. [#10242](#) ([Movses](#)).
- Added system tables for users, roles, grants, settings profiles, quotas, row policies; added commands `SHOW USER`, `SHOW [CURRENT|ENABLED] ROLES`, `SHOW SETTINGS PROFILES`. [#10387](#) ([Vitaly Baranov](#)).
- Support writes in ODBC Table function [#10554](#) ([ageraab](#)). [#10901](#) ([tavplubix](#)).
- Add query performance metrics based on Linux `perf_events` (these metrics are calculated with hardware CPU counters and OS counters). It is optional and requires `CAP_SYS_ADMIN` to be set on clickhouse binary. [#9545](#) [Andrey Skobtsov](#). [#11226](#) ([Alexander Kuzmenkov](#)).
- Now support `NULL` and `NOT NULL` modifiers for data types in `CREATE` query. [#11057](#) ([Павел Потемкин](#)).
- Add `ArrowStream` input and output format. [#11088](#) ([hc2](#)).
- Support Cassandra as external dictionary source. [#4978](#) ([favstovol](#)).
- Added a new layout `direct` which loads all the data directly from the source for each query, without storing or caching data. [#10622](#) ([Artem Streltsov](#)).
- Added new `complex_key_direct` layout to dictionaries, that does not store anything locally during query execution. [#10850](#) ([Artem Streltsov](#)).
- Added support for MySQL style global variables syntax (stub). This is needed for compatibility of MySQL protocol. [#11832](#) ([alexey-milovidov](#)).
- Added syntax highlighting to `clickhouse-client` using `replxx`. [#11422](#) ([Tagir Kuskarov](#)).
- `minMap` and `maxMap` functions were added. [#11603](#) ([Ildus Kurbangaliev](#)).
- Add the `system.asynchronous_metric_log` table that logs historical metrics from `system.asynchronous_metrics`. [#11588](#) ([Alexander Kuzmenkov](#)).
- Add functions `extractAllGroupsHorizontal(haystack, re)` and `extractAllGroupsVertical(haystack, re)`. [#11554](#) ([Vasily Nemkov](#)).
- Add `SHOW CLUSTER(S)` queries. [#11467](#) ([hexiaoting](#)).
- Add `netloc` function for extracting network location, similar to `urlparse(url).netloc` in python. [#11356](#) ([Guillaume Tassery](#)).
- Add 2 more virtual columns for `engine=Kafka` to access message headers. [#11283](#) ([filimonov](#)).
- Add `_timestamp_ms` virtual column for Kafka engine (type is `Nullable(DateTime64(3))`). [#11260](#) ([filimonov](#)).
- Add function `randomFixedString`. [#10866](#) ([Andrei Nekrashevich](#)).
- Add function `fuzzBits` that randomly flips bits in a string with given probability. [#11237](#) ([Andrei Nekrashevich](#)).

- Allow comparison of numbers with constant string in comparison operators, IN and VALUES sections. #11647 (alexey-milovidov).
- Add `round_robin` load\_balancing mode. #11645 (Azat Khuzhin).
- Add `cast_keep_nullable` setting. If set `CAST(something_nullable AS Type)` return `Nullable(Type)`. #11733 (Artem Zuikov).
- Added column position to `system.columns` table and `column_position` to `system.parts_columns` table. It contains ordinal position of a column in a table starting with 1. This closes #7744. #11655 (alexey-milovidov).
- ON CLUSTER support for SYSTEM {FLUSH DISTRIBUTED,STOP/START DISTRIBUTED SEND}. #11415 (Azat Khuzhin).
- Add `system.distribution_queue` table. #11394 (Azat Khuzhin).
- Support for all format settings in Kafka, expose some setting on table level, adjust the defaults for better performance. #11388 (filimonov).
- Add `port` function (to extract port from URL). #11120 (Azat Khuzhin).
- Now `dictGet*` functions accept table names. #11050 (Vitaly Baranov).
- The `clickhouse-format` tool is now able to format multiple queries when the `-n` argument is used. #10852 (Dario).
- Possibility to configure proxy-resolver for DiskS3. #10744 (Pavel Kovalenko).
- Make `pointInPolygon` work with non-constant polygon. `PointInPolygon` now can take `Array(Array(Tuple(..., ...)))` as second argument, array of polygon and holes. #10623 (Alexey Ilyukhov) #11421 (Alexey Ilyukhov).
- Added `move_ttl_info` to `system.parts` in order to provide introspection of move TTL functionality. #10591 (Vladimir Chebotarev).
- Possibility to work with S3 through proxies. #10576 (Pavel Kovalenko).
- Add `NCHAR` and `NVARCHAR` synonyms for data types. #11025 (alexey-milovidov).
- Resolved #7224: added `FailedQuery`, `FailedSelectQuery` and `FailedInsertQuery` metrics to `system.events` table. #11151 (Nikita Orlov).
- Add more `jemalloc` statistics to `system.asynchronous_metrics`, and ensure that we see up-to-date values for them. #11748 (Alexander Kuzmenkov).
- Allow to specify default S3 credentials and custom auth headers. #11134 (Grigory Pervakov).
- Added new functions to import/export `DateTime64` as `Int64` with various precision: `to-/fromUnixTimestamp64Milli/-Micro/-Nano`. #10923 (Vasily Nemkov).
- Allow specifying `mongodb://` URI for MongoDB dictionaries. #10915 (Alexander Kuzmenkov).
- `OFFSET` keyword can now be used without an affiliated `LIMIT` clause. #10802 (Guillaume Tassery).
- Added `system.licenses` table. This table contains licenses of third-party libraries that are located in `contrib` directory. This closes #2890. #10795 (alexey-milovidov).
- New function function `toStartOfSecond(DateTime64) -> DateTime64` that nullifies sub-second part of `DateTime64` value. #10722 (Vasily Nemkov).
- Add new input format `JSONAsString` that accepts a sequence of JSON objects separated by newlines, spaces and/or commas. #10607 (Kruglov Pavel).

- Allowed to profile memory with finer granularity steps than 4 MiB. Added sampling memory profiler to capture random allocations/deallocations. #10598 (alexey-milovidov).
- SimpleAggregateFunction now also supports sumMap. #10000 (Ildus Kurbangaliev).
- Support ALTER RENAME COLUMN for the distributed table engine. Continuation of #10727. Fixes #10747. #10887 (alesapin).

## Bug Fix

- Fix UBSan report in Decimal parse. This fixes #7540. #10512 (alexey-milovidov).
- Fix potential floating point exception when parsing DateTime64. This fixes #11374. #11875 (alexey-milovidov).
- Fix rare crash caused by using Nullable column in prewhere condition. #11895 #11608 #11869 (Nikolai Kochetov).
- Don't allow arrayJoin inside higher order functions. It was leading to broken protocol synchronization. This closes #3933. #11846 (alexey-milovidov).
- Fix wrong result of comparison of FixedString with constant String. This fixes #11393. This bug appeared in version 20.4. #11828 (alexey-milovidov).
- Fix wrong result for if with NULLs in condition. #11807 (Artem Zuikov).
- Fix using too many threads for queries. #11788 (Nikolai Kochetov).
- Fixed Scalar does not exist exception when using WITH <scalar subquery> ... in SELECT ... FROM merge\_tree\_table ... #11621. #11767 (Amos Bird).
- Fix unexpected behaviour of queries like SELECT \*, xyz.\* which were success while an error expected. #11753 (hexiaoting).
- Now replicated fetches will be cancelled during metadata alter. #11744 (alesapin).
- Parse metadata stored in zookeeper before checking for equality. #11739 (Azat Khuzhin).
- Fixed LOGICAL\_ERROR caused by wrong type deduction of complex literals in Values input format. #11732 (tavplubix).
- Fix ORDER BY ... WITH FILL over const columns. #11697 (Anton Popov).
- Fix very rare race condition in SYSTEM SYNC REPLICA. If the replicated table is created and at the same time from the separate connection another client is issuing SYSTEM SYNC REPLICA command on that table (this is unlikely, because another client should be aware that the table is created), it's possible to get nullptr dereference. #11691 (alexey-milovidov).
- Pass proper timeouts when communicating with XDBC bridge. Recently timeouts were not respected when checking bridge liveness and receiving meta info. #11690 (alexey-milovidov).
- Fix LIMIT n WITH TIES usage together with ORDER BY statement, which contains aliases. #11689 (Anton Popov).
- Fix possible Pipeline stuck for selects with parallel FINAL. Fixes #11636. #11682 (Nikolai Kochetov).
- Fix error which leads to an incorrect state of system.mutations. It may show that whole mutation is already done but the server still has MUTATE\_PART tasks in the replication queue and tries to execute them. This fixes #11611. #11681 (alesapin).
- Fix syntax hilite in CREATE USER query. #11664 (alexey-milovidov).

- Add support for regular expressions with case-insensitive flags. This fixes #11101 and fixes #11506. #11649 (alexey-milovidov).
- Remove trivial count query optimization if row-level security is set. In previous versions the user get total count of records in a table instead filtered. This fixes #11352. #11644 (alexey-milovidov).
- Fix bloom filters for String (data skipping indices). #11638 (Azat Khuzhin).
- Without -q option the database does not get created at startup. #11604 (giordyb).
- Fix error `Block structure mismatch` for queries with sampling reading from `Buffer` table. #11602 (Nikolai Kochetov).
- Fix wrong exit code of the clickhouse-client, when `exception.code() % 256 == 0`. #11601 (filimonov).
- Fix race conditions in CREATE/DROP of different replicas of ReplicatedMergeTree. Continue to work if the table was not removed completely from ZooKeeper or not created successfully. This fixes #11432. #11592 (alexey-milovidov).
- Fix trivial error in log message about "Mark cache size was lowered" at server startup. This closes #11399. #11589 (alexey-milovidov).
- Fix error `Size of offsets does not match size of column` for queries with `PREWHERE` column in (subquery) and `ARRAY JOIN`. #11580 (Nikolai Kochetov).
- Fixed rare segfault in `SHOW CREATE TABLE` Fixes #11490. #11579 (tavplubix).
- All queries in HTTP session have had the same `query_id`. It is fixed. #11578 (tavplubix).
- Now clickhouse-server docker container will prefer IPv6 checking server aliveness. #11550 (Ivan Starkov).
- Fix the error `Data compressed with different methods` that can happen if `min_bytes_to_use_direct_io` is enabled and `PREWHERE` is active and using SAMPLE or high number of threads. This fixes #11539. #11540 (alexey-milovidov).
- Fix shard\_num/replica\_num for `<node>` (breaks `use_compact_format_in_distributed_parts_names`). #11528 (Azat Khuzhin).
- Fix async INSERT into Distributed for `prefer_localhost_replica=0` and w/o internal\_replication. #11527 (Azat Khuzhin).
- Fix memory leak when exception is thrown in the middle of aggregation with `-State` functions. This fixes #8995. #11496 (alexey-milovidov).
- Fix Pipeline stuck exception for `INSERT SELECT FINAL` where `SELECT (max_threads>1)` has multiple streams but `INSERT` has only one (`max_insert_threads==0`). #11455 (Azat Khuzhin).
- Fix wrong result in queries like `select count() from t, u`. #11454 (Artem Zuikov).
- Fix return compressed size for codecs. #11448 (Nikolai Kochetov).
- Fix server crash when a column has compression codec with non-literal arguments. Fixes #11365. #11431 (alesapin).
- Fix potential uninitialized memory read in MergeTree shutdown if table was not created successfully. #11420 (alexey-milovidov).
- Fix crash in JOIN over `LowCardinality(T)` and `Nullable(T)`. #11380. #11414 (Artem Zuikov).
- Fix error code for wrong `USING` key. #11373. #11404 (Artem Zuikov).

- Fixed `geohashesInBox` with arguments outside of latitude/longitude range. #11403 (Vasily Nemkov).
- Better errors for `joinGet()` functions. #11389 (Artem Zuikov).
- Fix possible Pipeline stuck error for queries with external sort and limit. Fixes #11359. #11366 (Nikolai Kochetov).
- Remove redundant lock during parts send in ReplicatedMergeTree. #11354 (alesapin).
- Fix support for `\G` (vertical output) in clickhouse-client in multiline mode. This closes #9933. #11350 (alexey-milovidov).
- Fix potential segfault when using `Lazy` database. #11348 (alexey-milovidov).
- Fix crash in direct selects from `Join` table engine (without JOIN) and wrong nullability. #11340 (Artem Zuikov).
- Fix crash in `quantilesExactWeightedArray`. #11337 (Nikolai Kochetov).
- Now merges stopped before change metadata in `ALTER` queries. #11335 (alesapin).
- Make writing to `MATERIALIZED VIEW` with setting `parallel_view_processing = 1` parallel again. Fixes #10241. #11330 (Nikolai Kochetov).
- Fix `visitParamExtractRaw` when extracted JSON has strings with unbalanced { or [. #11318 (Ewout).
- Fix very rare race condition in ThreadPool. #11314 (alexey-milovidov).
- Fix insignificant data race in `clickhouse-copier`. Found by integration tests. #11313 (alexey-milovidov).
- Fix potential uninitialized memory in conversion. Example: `SELECT toIntervalSecond(now64())`. #11311 (alexey-milovidov).
- Fix the issue when index analysis cannot work if a table has Array column in primary key and if a query is filtering by this column with `empty` or `notEmpty` functions. This fixes #11286. #11303 (alexey-milovidov).
- Fix bug when query speed estimation can be incorrect and the limit of `min_execution_speed` may not work or work incorrectly if the query is throttled by `max_network_bandwidth`, `max_execution_speed` or `priority` settings. Change the default value of `timeout_before_checking_execution_speed` to non-zero, because otherwise the settings `min_execution_speed` and `max_execution_speed` have no effect. This fixes #11297. This fixes #5732. This fixes #6228. Usability improvement: avoid concatenation of exception message with progress bar in `clickhouse-client`. #11296 (alexey-milovidov).
- Fix crash when `SET DEFAULT ROLE` is called with wrong arguments. This fixes #10586. #11278 (Vitaly Baranov).
- Fix crash while reading malformed data in Protobuf format. This fixes #5957, fixes #11203. #11258 (Vitaly Baranov).
- Fixed a bug when `cache dictionary` could return default value instead of normal (when there are only expired keys). This affects only string fields. #11233 (Nikita Mikhaylov).
- Fix error `Block structure mismatch` in `QueryPipeline` while reading from `VIEW` with constants in inner query. Fixes #11181. #11205 (Nikolai Kochetov).
- Fix possible exception `Invalid status for associated output` #11200 (Nikolai Kochetov).
- Now `primary.idx` will be checked if it's defined in `CREATE` query. #11199 (alesapin).

- Fix possible error `Cannot capture column` for higher-order functions with `Array(Array(LowCardinality))` captured argument. [#11185 \(Nikolai Kochetov\)](#).
- Fixed S3 globbing which could fail in case of more than 1000 keys and some backends. [#11179 \(Vladimir Chebotarev\)](#).
- If data skipping index is dependent on columns that are going to be modified during background merge (for SummingMergeTree, AggregatingMergeTree as well as for TTL GROUP BY), it was calculated incorrectly. This issue is fixed by moving index calculation after merge so the index is calculated on merged data. [#11162 \(Azat Khuzhin\)](#).
- Fix for the hang which was happening sometimes during DROP of table engine=Kafka (or during server restarts). [#11145 \(filimonov\)](#).
- Fix excessive reserving of threads for simple queries (optimization for reducing the number of threads, which was partly broken after changes in pipeline). [#11114 \(Azat Khuzhin\)](#).
- Remove logging from mutation finalization task if nothing was finalized. [#11109 \(alesapin\)](#).
- Fixed deadlock during server startup after update with changes in structure of system log tables. [#11106 \(alesapin\)](#).
- Fixed memory leak in `registerDiskS3`. [#11074 \(Pavel Kovalenko\)](#).
- Fix error `No such name in Block::erase()` when JOIN appears with PREWHERE or `optimize_move_to_prewhere` makes PREWHERE from WHERE. [#11051 \(Artem Zuikov\)](#).
- Fixes the potential missed data during termination of Kafka engine table. [#11048 \(filimonov\)](#).
- Fixed `parseDateTime64BestEffort` argument resolution bugs. [#10925](#). [#11038 \(Vasily Nemkov\)](#).
- Now it's possible to `ADD/DROP` and `RENAME` the same one column in a single `ALTER` query. Exception message for simultaneous `MODIFY` and `RENAME` became more clear. Partially fixes [#10669](#). [#11037 \(alesapin\)](#).
- Fixed parsing of S3 URLs. [#11036 \(Vladimir Chebotarev\)](#).
- Fix memory tracking for two-level `GROUP BY` when there is a `LIMIT`. [#11022 \(Azat Khuzhin\)](#).
- Fix very rare potential use-after-free error in MergeTree if table was not created successfully. [#10986 \(alexey-milovidov\)](#).
- Fix metadata (relative path for rename) and data (relative path for symlink) handling for Atomic database. [#10980 \(Azat Khuzhin\)](#).
- Fix server crash on concurrent `ALTER` and `DROP DATABASE` queries with Atomic database engine. [#10968 \(tavplubix\)](#).
- Fix incorrect raw data size in method `getRawData()`. [#10964 \(lgr\)](#).
- Fix incompatibility of two-level aggregation between versions 20.1 and earlier. This incompatibility happens when different versions of ClickHouse are used on initiator node and remote nodes and the size of GROUP BY result is large and aggregation is performed by a single String field. It leads to several unmerged rows for a single key in result. [#10952 \(alexey-milovidov\)](#).
- Avoid sending partially written files by the `DistributedBlockOutputStream`. [#10940 \(Azat Khuzhin\)](#).
- Fix crash in `SELECT count(notNullIn(NULL, []))`. [#10920 \(Nikolai Kochetov\)](#).
- Fix for the hang which was happening sometimes during DROP of table engine=Kafka (or during server restarts). [#10910 \(filimonov\)](#).

- Now it's possible to execute multiple `ALTER RENAME` like `a TO b, c TO a`. #10895 (alesapin).
- Fix possible race which could happen when you get result from aggregate function state from multiple thread for the same column. The only way (which I found) it can happen is when you use `finalizeAggregation` function while reading from table with `Memory` engine which stores `AggregateFunction` state for `quanite*` function. #10890 (Nikolai Kochetov).
- Fix backward compatibility with tuples in Distributed tables. #10889 (Anton Popov).
- Fix SIGSEGV in `StringHashTable` (if such key does not exist). #10870 (Azat Khuzhin).
- Fixed `WATCH` hangs after `LiveView` table was dropped from database with `Atomic` engine. #10859 (tavplubix).
- Fixed bug in `ReplicatedMergeTree` which might cause some `ALTER` on `OPTIMIZE` query to hang waiting for some replica after it become inactive. #10849 (tavplubix).
- Now constraints are updated if the column participating in `CONSTRAINT` expression was renamed. Fixes #10844. #10847 (alesapin).
- Fix potential read of uninitialized memory in cache dictionary. #10834 (alexey-milovidov).
- Fix columns order after `Block::sortColumns()` (also add a test that shows that it affects some real use case - Buffer engine). #10826 (Azat Khuzhin).
- Fix the issue with ODBC bridge when no quoting of identifiers is requested. This fixes #7984. #10821 (alexey-milovidov).
- Fix UBSan and MSan report in DateLUT. #10798 (alexey-milovidov).
- Make use of `src_type` for correct type conversion in key conditions. Fixes #6287. #10791 (Andrew Onyshchuk).
- Get rid of old libunwind patches. <https://github.com/ClickHouse-Extras/libunwind/commit/500aa227911bd185a94bfc071d68f4d3b03cb3b1#r39048012> This allows to disable `-fno-omit-frame-pointer` in `clang` builds that improves performance at least by 1% in average. #10761 (Amos Bird).
- Fix avgWeighted when using floating-point weight over multiple shards. #10758 (Baudouin Giard).
- Fix `parallel_view_processing` behavior. Now all insertions into `MATERIALIZED VIEW` without exception should be finished if exception happened. Fixes #10241. #10757 (Nikolai Kochetov).
- Fix combinator `-OrNull` and `-OrDefault` when combined with `-State`. #10741 (hcz).
- 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 usage of primary key wrapped into a function with 'FINAL' modifier and 'ORDER BY' optimization. #10715 (Anton Popov).
- Fix possible buffer overflow in function `h3EdgeAngle`. #10711 (alexey-milovidov).
- Fix disappearing totals. Totals could have been filtered if query had had join or subquery with external where condition. Fixes #10674. #10698 (Nikolai Kochetov).
- Fix atomicity of HTTP insert. This fixes #9666. #10687 (Andrew Onyshchuk).
- Fix multiple usages of `IN` operator with the identical set in one query. #10686 (Anton Popov).

- Fixed bug, which causes http requests stuck on client close when `readonly=2` and `cancel_http_READONLY_queries_on_client_close=1`. Fixes #7939, #7019, #7736, #7091. #10684 (tavplubix).
- Fix order of parameters in `AggregateTransform` constructor. #10667 (palasonic1).
- 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 bug which locks concurrent alters when table has a lot of parts. #10659 (alesapin).
- Fix nullptr dereference in `StorageBuffer` if server was shutdown before table startup. #10641 (alexey-milovidov).
- Fix predicates optimization for distributed queries (`enable_optimize_predicate_expression=1`) for queries with `HAVING` section (i.e. when filtering on the server initiator is required), by preserving the order of expressions (and this is enough to fix), and also force aggregator use column names over indexes. Fixes: #10613, #11413. #10621 (Azat Khuzhin).
- Fix `optimize_skip_unused_shards` with `LowCardinality`. #10611 (Azat Khuzhin).
- Fix segfault in `StorageBuffer` when exception on server startup. Fixes #10550. #10609 (tavplubix).
- On `SYSTEM DROP DNS CACHE` query also drop caches, which are used to check if user is allowed to connect from some IP addresses. #10608 (tavplubix).
- Fixed incorrect scalar results inside inner query of `MATERIALIZED VIEW` in case if this query contained dependent table. #10603 (Nikolai Kochetov).
- Fixed handling condition variable for synchronous mutations. In some cases signals to that condition variable could be lost. #10588 (Vladimir Chebotarev).
- Fixes possible crash `createDictionary()` is called before `loadStoredObject()` has finished. #10587 (Vitaly Baranov).
- Fix error `the BloomFilter false positive must be a double number between 0 and 1` #10551. #10569 (Winter Zhang).
- Fix `SELECT` of column `ALIAS` which default expression type different from column type. #10563 (Azat Khuzhin).
- Implemented comparison between `DateTime64` and `String` values (just like for `DateTime`). #10560 (Vasily Nemkov).
- Fix index corruption, which may occur in some cases after merge compact parts into another compact part. #10531 (Anton Popov).
- Disable `GROUP BY sharding_key` optimization by default (`optimize_distributed_group_by_sharding_key` had been introduced and turned off by default, due to trickery of `sharding_key` analyzing, simple example is `if` in `sharding key`) and fix it for `WITH ROLLUP/CUBE/TOTALS`. #10516 (Azat Khuzhin).
- Fixes: #10263 (after that PR dist send via `INSERT` had been postponing on each `INSERT`) Fixes: #8756 (that PR breaks distributed sends with all of the following conditions met (unlikely setup for now I guess): `internal_replication == false`, multiple local shards (activates the hardlinking code) and `distributed_storage_policy` (makes `link(2)` fails on `EXDEV`)). #10486 (Azat Khuzhin).
- Fixed error with "max\_rows\_to\_sort" limit. #10268 (alexey-milovidov).

- Get dictionary and check access rights only once per each call of any function reading external dictionaries. #10928 (Vitaly Baranov).

## Improvement

- Apply TTL for old data, after ALTER MODIFY TTL query. This behaviour is controlled by setting materialize\_ttl\_after\_modify, which is enabled by default. #11042 (Anton Popov).
- When parsing C-style backslash escapes in string literals, VALUES and various text formats (this is an extension to SQL standard that is endemic for ClickHouse and MySQL), keep backslash if unknown escape sequence is found (e.g. \% or \w) that will make usage of LIKE and match regular expressions more convenient (it's enough to write name LIKE 'used\\_cars' instead of name LIKE 'used\\\_cars') and more compatible at the same time. This fixes #10922. #11208 (alexey-milovidov).
- When reading Decimal value, cut extra digits after point. This behaviour is more compatible with MySQL and PostgreSQL. This fixes #10202. #11831 (alexey-milovidov).
- Allow to DROP replicated table if the metadata in ZooKeeper was already removed and does not exist (this is also the case when using TestKeeper for testing and the server was restarted). Allow to RENAME replicated table even if there is an error communicating with ZooKeeper. This fixes #10720. #11652 (alexey-milovidov).
- Slightly improve diagnostic of reading decimal from string. This closes #10202. #11829 (alexey-milovidov).
- Fix sleep invocation in signal handler. It was sleeping for less amount of time than expected. #11825 (alexey-milovidov).
- (Only Linux) OS related performance metrics (for CPU and I/O) will work even without CAP\_NET\_ADMIN capability. #10544 (Alexander Kazakov).
- Added hostname as an alias to function hostName. This feature was suggested by Victor Tarnavskiy from Yandex.Metrica. #11821 (alexey-milovidov).
- Added support for distributed DDL (update/delete/drop partition) on cross replication clusters. #11703 (Nikita Mikhaylov).
- Emit warning instead of error in server log at startup if we cannot listen one of the listen addresses (e.g. IPv6 is unavailable inside Docker). Note that if server fails to listen all listed addresses, it will refuse to startup as before. This fixes #4406. #11687 (alexey-milovidov).
- Default user and database creation on docker image starting. #10637 (Paramtamtam).
- When multiline query is printed to server log, the lines are joined. Make it to work correct in case of multiline string literals, identifiers and single-line comments. This fixes #3853. #11686 (alexey-milovidov).
- Multiple names are now allowed in commands: CREATE USER, CREATE ROLE, ALTER USER, SHOW CREATE USER, SHOW GRANTS and so on. #11670 (Vitaly Baranov).
- Add support for distributed DDL (UPDATE/DELETE/DROP PARTITION) on cross replication clusters. #11508 (frank lee).
- Clear password from command line in clickhouse-client and clickhouse-benchmark if the user has specified it with explicit value. This prevents password exposure by ps and similar tools. #11665 (alexey-milovidov).
- Don't use debug info from ELF file if it does not correspond to the running binary. It is needed to avoid printing wrong function names and source locations in stack traces. This fixes #7514. #11657 (alexey-milovidov).

- Return NULL/zero when value is not parsed completely in `parseDateTimeBestEffortOrNull`/Zero functions. This fixes #7876. #11653 (alexey-milovidov).
- Skip empty parameters in requested URL. They may appear when you write `http://localhost:8123/?&a=b` or `http://localhost:8123/?a=b&&c=d`. This closes #10749. #11651 (alexey-milovidov).
- Allow using `groupArrayArray` and `groupUniqArrayArray` as `SimpleAggregateFunction`. #11650 (Volodymyr Kuznetsov).
- Allow comparison with constant strings by implicit conversions when analysing index conditions on other types. This may close #11630. #11648 (alexey-milovidov).
- <https://github.com/ClickHouse/ClickHouse/pull/7572#issuecomment-642815377> Support config default HTTPHandlers. #11628 (Winter Zhang).
- Make more input formats to work with Kafka engine. Fix the issue with premature flushes. Fix the performance issue when `kafka_num_consumers` is greater than number of partitions in topic. #11599 (filimonov).
- Improve `multiple_joins_rewriter_version=2` logic. Fix unknown columns error for lambda aliases. #11587 (Artem Zuikov).
- Better exception message when cannot parse columns declaration list. This closes #10403. #11537 (alexey-milovidov).
- Improve `enable_optimize_predicate_expression=1` logic for VIEW. #11513 (Artem Zuikov).
- Adding support for PREWHERE in live view tables. #11495 (vzakaznikov).
- Automatically update DNS cache, which is used to check if user is allowed to connect from an address. #11487 (tavplubix).
- OPTIMIZE FINAL will force merge even if concurrent merges are performed. This closes #11309 and closes #11322. #11346 (alexey-milovidov).
- Suppress output of cancelled queries in clickhouse-client. In previous versions result may continue to print in terminal even after you press Ctrl+C to cancel query. This closes #9473. #11342 (alexey-milovidov).
- Now history file is updated after each query and there is no race condition if multiple clients use one history file. This fixes #9897. #11453 (Tagir Kuskarov).
- Better log messages in while reloading configuration. #11341 (alexey-milovidov).
- Remove trailing whitespaces from formatted queries in `clickhouse-client` or `clickhouse-format` in some cases. #11325 (alexey-milovidov).
- Add setting "output\_format\_pretty\_max\_value\_width". If value is longer, it will be cut to avoid output of too large values in terminal. This closes #11140. #11324 (alexey-milovidov).
- Better exception message in case when there is shortage of memory mappings. This closes #11027. #11316 (alexey-milovidov).
- Support (U)Int8, (U)Int16, Date in ASOF JOIN. #11301 (Artem Zuikov).
- Support `kafka_client_id` parameter for Kafka tables. It also changes the default `client.id` used by ClickHouse when communicating with Kafka to be more verbose and usable. #11252 (filimonov).
- Keep the value of `DistributedFilesToInsert` metric on exceptions. In previous versions, the value was set when we are going to send some files, but it is zero, if there was an exception and some files are still pending. Now it corresponds to the number of pending files in filesystem. #11220 (alexey-milovidov).

- Add support for multi-word data type names (such as `DOUBLE PRECISION` and `CHAR VARYING`) for better SQL compatibility. #11214 (Павел Потемкин).
- Provide synonyms for some data types. #10856 (Павел Потемкин).
- The query log is now enabled by default. #11184 (Ivan Blinkov).
- Show authentication type in table `system.users` and while executing `SHOW CREATE USER` query. #11080 (Vitaly Baranov).
- Remove data on explicit `DROP DATABASE` for `Memory` database engine. Fixes #10557. #11021 (tavplubix).
- Set thread names for internal threads of rdkafka library. Make logs from rdkafka available in server logs. #10983 (Azat Khuzhin).
- Support for unicode whitespaces in queries. This helps when queries are copy-pasted from Word or from web page. This fixes #10896. #10903 (alexey-milovidov).
- Allow large `UInt` types as the index in function `tupleElement`. #10874 (hcz).
- Respect `prefer_localhost_replica/load_balancing` on `INSERT` into `Distributed`. #10867 (Azat Khuzhin).
- Introduce `min_insert_block_size_rows_for_materialized_views`, `min_insert_block_size_bytes_for_materialized_views` settings. These settings are similar to `min_insert_block_size_rows` and `min_insert_block_size_bytes`, but applied only for blocks inserted into `MATERIALIZED VIEW`. It helps to control blocks squashing while pushing to MVs and avoid excessive memory usage. #10858 (Azat Khuzhin).
- Get rid of exception from replicated queue during server shutdown. Fixes #10819. #10841 (alesapin).
- Ensure that `varSamp`, `varPop` cannot return negative results due to numerical errors and that `stddevSamp`, `stddevPop` cannot be calculated from negative variance. This fixes #10532. #10829 (alexey-milovidov).
- Better DNS exception message. This fixes #10813. #10828 (alexey-milovidov).
- Change HTTP response code in case of some parse errors to 400 Bad Request. This fix #10636. #10640 (alexey-milovidov).
- Print a message if `clickhouse-client` is newer than `clickhouse-server`. #10627 (alexey-milovidov).
- Adding support for `INSERT INTO [db.]table WATCH` query. #10498 (vzakaznikov).
- Allow to pass `quota_key` in `clickhouse-client`. This closes #10227. #10270 (alexey-milovidov).

## Performance Improvement

- Allow multiple replicas to assign merges, mutations, partition drop, move and replace concurrently. This closes #10367. #11639 (alexey-milovidov) #11795 (alexey-milovidov).
- Optimization of GROUP BY with respect to table sorting key, enabled with `optimize_aggregation_in_order` setting. #9113 (dimarub2000).
- Selects with final are executed in parallel. Added setting `max_final_threads` to limit the number of threads used. #10463 (Nikolai Kochetov).
- Improve performance for `INSERT` queries via `INSERT SELECT` or `INSERT` with `clickhouse-client` when small blocks are generated (typical case with parallel parsing). This fixes #11275. Fix the issue that `CONSTRAINTs` were not working for `DEFAULT` fields. This fixes #11273. Fix the issue that `CONSTRAINTs` were ignored for `TEMPORARY` tables. This fixes #11274. #11276 (alexey-milovidov).
- Optimization that eliminates min/max/any aggregators of GROUP BY keys in SELECT section, enabled with `optimize_aggregators_of_group_by_keys` setting. #11667 (xPoSx). #11806 (Azat Khuzhin).

- New optimization that takes all operations out of `any` function, enabled with `optimize_move_functions_out_of_any #11529` ([Ruslan](#)).
- Improve performance of `clickhouse-client` in interactive mode when Pretty formats are used. In previous versions, significant amount of time can be spent calculating visible width of UTF-8 string. This closes [#11323](#). [#11323](#) ([alexey-milovidov](#)).
- Improved performance for queries with `ORDER BY` and small `LIMIT` (less, then `max_block_size`). [#11171](#) ([Albert Kidrachev](#)).
- Add runtime CPU detection to select and dispatch the best function implementation. Add support for codegeneration for multiple targets. This closes [#1017](#). [#10058](#) ([DimasKovas](#)).
- Enable `mlock` of clickhouse binary by default. It will prevent clickhouse executable from being paged out under high IO load. [#11139](#) ([alexey-milovidov](#)).
- Make queries with `sum` aggregate function and without GROUP BY keys to run multiple times faster. [#10992](#) ([alexey-milovidov](#)).
- Improving radix sort (used in `ORDER BY` with simple keys) by removing some redundant data moves. [#10981](#) ([Arslan Gumerov](#)).
- Sort bigger parts of the left table in MergeJoin. Buffer left blocks in memory. Add `partial_merge_join_left_table_buffer_bytes` setting to manage the left blocks buffers sizes. [#10601](#) ([Artem Zuikov](#)).
- Remove duplicate `ORDER BY` and `DISTINCT` from subqueries, this optimization is enabled with `optimize_duplicate_order_by_and_distinct #10067` ([Mikhail Malafeev](#)).
- This feature eliminates functions of other keys in GROUP BY section, enabled with `optimize_group_by_function_keys #10051` ([xPoSx](#)).
- New optimization that takes arithmetic operations out of aggregate functions, enabled with `optimize_arithmetic_operations_in_aggregate_functions #10047` ([Ruslan](#)).
- Use HTTP client for S3 based on Poco instead of curl. This will improve performance and lower memory usage of s3 storage and table functions. [#11230](#) ([Pavel Kovalenko](#)).
- Fix Kafka performance issue related to reschedules based on limits, which were always applied. [#11149](#) ([filimonov](#)).
- Enable `percpu_arena:percpu` for jemalloc (This will reduce memory fragmentation due to thread pool). [#11084](#) ([Azat Khuzhin](#)).
- Optimize memory usage when reading a response from an S3 HTTP client. [#11561](#) ([Pavel Kovalenko](#)).
- Adjust the default Kafka settings for better performance. [#11388](#) ([filimonov](#)).

## Experimental Feature

- Add data type `Point` (`Tuple(Float64, Float64)`) and `Polygon` (`Array(Array(Tuple(Float64, Float64)))`). [#10678](#) ([Alexey Ilyukhov](#)).
- Add's a `hasSubstr` function that allows for look for subsequences in arrays. Note: this function is likely to be renamed without further notice. [#11071](#) ([Ryad Zenine](#)).

- Added OpenCL support and bitonic sort algorithm, which can be used for sorting integer types of data in single column. Needs to be build with flag `-DENABLE_OPENCL=1`. For using bitonic sort algorithm instead of others you need to set `bitonic_sort` for Setting's option `special_sort` and make sure that OpenCL is available. This feature does not improve performance or anything else, it is only provided as an example and for demonstration purposes. It is likely to be removed in near future if there will be no further development in this direction. [#10232 \(Ri\)](#).

## Build/Testing/Packaging Improvement

- Enable clang-tidy for programs and utils. [#10991 \(alexey-milovidov\)](#).
- Remove dependency on `tzdata`: do not fail if `/usr/share/zoneinfo` directory does not exist. Note that all timezones work in ClickHouse even without tzdata installed in system. [#11827 \(alexey-milovidov\)](#).
- Added MSan and UBSan stress tests. Note that we already have MSan, UBSan for functional tests and "stress" test is another kind of tests. [#10871 \(alexey-milovidov\)](#).
- Print compiler build id in crash messages. It will make us slightly more certain about what binary has crashed. Added new function `buildId`. [#11824 \(alexey-milovidov\)](#).
- Added a test to ensure that mutations continue to work after FREEZE query. [#11820 \(alexey-milovidov\)](#).
- Don't allow tests with "fail" substring in their names because it makes looking at the tests results in browser less convenient when you type Ctrl+F and search for "fail". [#11817 \(alexey-milovidov\)](#).
- Removes unused imports from `HTTPHandlerFactory`. [#11660 \(Bharat Nallan\)](#).
- Added a random sampling of instances where copier is executed. It is needed to avoid `Too many simultaneous queries` error. Also increased timeout and decreased fault probability. [#11573 \(Nikita Mikhaylov\)](#).
- Fix missed include. [#11525 \(Matvey V. Kornilov\)](#).
- Speed up build by removing old example programs. Also found some orphan functional test. [#11486 \(alexey-milovidov\)](#).
- Increase ccache size for builds in CI. [#11450 \(alesapin\)](#).
- Leave only `unit_tests_dbms` in deb build. [#11429 \(Ilya Yatsishin\)](#).
- Update librdkafka to version [1.4.2](#). [#11256 \(filimonov\)](#).
- Refactor CMake build files. [#11390 \(Ivan\)](#).
- Fix several flaky integration tests. [#11355 \(alesapin\)](#).
- Add support for unit tests run with UBSan. [#11345 \(alexey-milovidov\)](#).
- Remove redundant timeout from integration test `test_insertion_sync_fails_with_timeout`. [#11343 \(alesapin\)](#).
- Better check for hung queries in `clickhouse-test`. [#11321 \(alexey-milovidov\)](#).
- Emit a warning if server was build in debug or with sanitizers. [#11304 \(alexey-milovidov\)](#).
- Now `clickhouse-test` check the server aliveness before tests run. [#11285 \(alesapin\)](#).
- Fix potentially flacky test `00731_long_merge_tree_select_opened_files.sh`. It does not fail frequently but we have discovered potential race condition in this test while experimenting with ThreadFuzzer: [#9814](#) See link for the example. [#11270 \(alexey-milovidov\)](#).
- Repeat test in CI if `curl` invocation was timed out. It is possible due to system hangups for 10+ seconds that are typical in our CI infrastructure. This fixes [#11267](#). [#11268 \(alexey-milovidov\)](#).

- Add a test for Join table engine from @donmikel. This closes #9158. #11265 (alexey-milovidov).
- Fix several non significant errors in unit tests. #11262 (alesapin).
- Now parts of linker command for cctz library will not be shuffled with other libraries. #11213 (alesapin).
- Split /programs/server into actual program and library. #11186 (Ivan).
- Improve build scripts for protobuf & gRPC. #11172 (Vitaly Baranov).
- Enable performance test that was not working. #11158 (alexey-milovidov).
- Create root S3 bucket for tests before any CH instance is started. #11142 (Pavel Kovalenko).
- Add performance test for non-constant polygons. #11141 (alexey-milovidov).
- Fixing 00979\_live\_view\_watch\_continuous\_aggregates test. #11024 (vzakaznikov).
- Add ability to run zookeeper in integration tests over tmpfs. #11002 (alesapin).
- Wait for odbc-bridge with exponential backoff. Previous wait time of 200 ms was not enough in our CI environment. #10990 (alexey-milovidov).
- Fix non-deterministic test. #10989 (alexey-milovidov).
- Added a test for empty external data. #10926 (alexey-milovidov).
- Database is recreated for every test. This improves separation of tests. #10902 (alexey-milovidov).
- Added more asserts in columns code. #10833 (alexey-milovidov).
- Better cooperation with sanitizers. Print information about query\_id in the message of sanitizer failure. #10832 (alexey-milovidov).
- Fix obvious race condition in "Split build smoke test" check. #10820 (alexey-milovidov).
- Fix (false) MSan report in MergeTreeIndexFullText. The issue first appeared in #9968. #10801 (alexey-milovidov).
- Add MSan suppression for MariaDB Client library. #10800 (alexey-milovidov).
- GRPC make couldn't find protobuf files, changed make file by adding the right link. #10794 (mnkonkova).
- Enable extra warnings (-Weverything) for base, utils, programs. Note that we already have it for the most of the code. #10779 (alexey-milovidov).
- Suppressions of warnings from libraries was mistakenly declared as public in #10396. #10776 (alexey-milovidov).
- Restore a patch that was accidentally deleted in #10396. #10774 (alexey-milovidov).
- Fix performance tests errors, part 2. #10773 (alexey-milovidov).
- Fix performance test errors. #10766 (alexey-milovidov).
- Update cross-builds to use clang-10 compiler. #10724 (Ivan).
- Update instruction to install RPM packages. This was suggested by Denis (TG login @ldviolet) and implemented by Arkady Shejn. #10707 (alexey-milovidov).
- Trying to fix tests/queries/0\_stateless/01246\_insert\_into\_watch\_live\_view.py test. #10670 (vzakaznikov).

- Fixing and re-enabling 00979\_live\_view\_watch\_continuous\_aggregates.py test. #10658 (vzakaznikov).
- Fix OOM in ASan stress test. #10646 (alexey-milovidov).
- Fix UBSan report (adding zero to nullptr) in HashTable that appeared after migration to clang-10. #10638 (alexey-milovidov).
- Remove external call to `ld` (bfd) linker during tzdata processing in compile time. #10634 (alesapin).
- Allow to use `lld` to link blobs (resources). #10632 (alexey-milovidov).
- Fix UBSan report in LZ4 library. #10631 (alexey-milovidov). See also <https://github.com/lz4/lz4/issues/857>
- Update LZ4 to the latest dev branch. #10630 (alexey-milovidov).
- Added auto-generated machine-readable file with the list of stable versions. #10628 (alexey-milovidov).
- Fix `capnproto` version check for `capnp::UnalignedFlatArrayMessageReader`. #10618 (Matvey V. Kornilov).
- Lower memory usage in tests. #10617 (alexey-milovidov).
- Fixing hard coded timeouts in new live view tests. #10604 (vzakaznikov).
- Increasing timeout when opening a client in `tests/queries/0_stateless/helpers/client.py`. #10599 (vzakaznikov).
- Enable ThinLTO for clang builds, continuation of #10435. #10585 (Amos Bird).
- Adding fuzzers and preparing for oss-fuzz integration. #10546 (kyprizel).
- Fix FreeBSD build. #10150 (Ivan).
- Add new build for query tests using pytest framework. #10039 (Ivan).

## ClickHouse release v20.4

### ClickHouse release v20.4.8.99-stable 2020-08-10

#### Bug Fix

- Fixed error in `parseDateTimeBestEffort` function when unix timestamp was passed as an argument. This fixes #13362. #13441 (alexey-milovidov).
- Fixed potentially low performance and slightly incorrect result for `uniqExact`, `topK`, `sumDistinct` and similar aggregate functions called on `Float` types with `Nan` values. It also triggered assert in debug build. This fixes #12491. #13254 (alexey-milovidov).
- Fixed function `if` with nullable `constexpr` as cond that is not literal `NULL`. Fixes #12463. #13226 (alexey-milovidov).
- Fixed assert in `arrayElement` function in case of array elements are `Nullable` and array subscript is also `Nullable`. This fixes #12172. #13224 (alexey-milovidov).
- Fixed wrong index analysis with functions. It could lead to pruning wrong parts, while reading from `MergeTree` tables. Fixes #13060. Fixes #12406. #13081 (Anton Popov).
- Fixed unnecessary limiting for the number of threads for selects from local replica. #12840 (Nikolai Kochetov).
- Fixed possible extra overflow row in data which could appear for queries `WITH TOTALS`. #12747 (Nikolai Kochetov).

- Fixed performance with large tuples, which are interpreted as functions in `IN` section. The case when user write `WHERE x IN tuple(1, 2, ...)` instead of `WHERE x IN (1, 2, ...)` for some obscure reason. #12700 (Anton Popov).
- Fixed memory tracking for `input_format_parallel_parsing` (by attaching thread to group). #12672 (Azat Khuzhin).
- Fixed #12293 allow push predicate when subquery contains with clause. #12663 (Winter Zhang).
- Fixed #10572 fix bloom filter index with const expression. #12659 (Winter Zhang).
- Fixed `SIGSEGV` in `StorageKafka` when broker is unavailable (and not only). #12658 (Azat Khuzhin).
- Added support for function `if` with `Array(UUID)` arguments. This fixes #11066. #12648 (alexey-milovidov).
- Fixed race condition in external dictionaries with cache layout which can lead server crash. #12566 (alesapin).
- Removed data for Distributed tables (blocks from async INSERTs) on `DROP TABLE`. #12556 (Azat Khuzhin).
- Fixed bug which lead to broken old parts after `ALTER DELETE` query when `enable_mixed_granularity_parts=1`. Fixes #12536. #12543 (alesapin).
- Better exception for function `in` with invalid number of arguments. #12529 (Anton Popov).
- Fixed performance issue, while reading from compact parts. #12492 (Anton Popov).
- Fixed crash in JOIN with dictionary when we are joining over expression of dictionary key: `t JOIN dict ON expr(dict.id) = t.id`. Disable dictionary join optimisation for this case. #12458 (Artem Zuikov).
- Fixed possible segfault if StorageMerge. Closes #12054. #12401 (tavplubix).
- Fixed order of columns in `WITH FILL` modifier. Previously order of columns of `ORDER BY` statement wasn't respected. #12306 (Anton Popov).
- Avoid "bad cast" exception when there is an expression that filters data by virtual columns (like `_table` in Merge tables) or by "index" columns in system tables such as filtering by database name when querying from `system.tables`, and this expression returns `Nullable` type. This fixes #12166. #12305 (alexey-milovidov).
- Show error after TrieDictionary failed to load. #12290 (Vitaly Baranov).
- The function `arrayFill` worked incorrectly for empty arrays that may lead to crash. This fixes #12263. #12279 (alexey-milovidov).
- Implemented conversions to the common type for `LowCardinality` types. This allows to execute UNION ALL of tables with columns of `LowCardinality` and other columns. This fixes #8212. This fixes #4342. #12275 (alexey-milovidov).
- Fixed the behaviour when during multiple sequential inserts in `StorageFile` header for some special types was written more than once. This fixed #6155. #12197 (Nikita Mikhaylov).
- Fixed logical functions for `UInt8` values when they are not equal to 0 or 1. #12196 (Alexander Kazakov).
- Cap `max_memory_usage*` limits to the process resident memory. #12182 (Azat Khuzhin).
- Fixed `dictGet` arguments check during GROUP BY injective functions elimination. #12179 (Azat Khuzhin).
- Don't split the dictionary source's table name into schema and table name itself if ODBC connection does not support schema. #12165 (Vitaly Baranov).

- Fixed wrong logic in `ALTER DELETE` that leads to deleting of records when condition evaluates to NULL. This fixes #9088. This closes #12106. #12153 (alexey-milovidov).
- Fixed transform of query to send to external DBMS (e.g. MySQL, ODBC) in presence of aliases. This fixes #12032. #12151 (alexey-milovidov).
- Fixed potential overflow in integer division. This fixes #12119. #12140 (alexey-milovidov).
- Fixed potential infinite loop in `greatCircleDistance`, `geoDistance`. This fixes #12117. #12137 (alexey-milovidov).
- Normalize "pid" file handling. In previous versions the server may refuse to start if it was killed without proper shutdown and if there is another process that has the same pid as previously runned server. Also pid file may be removed in unsuccessful server startup even if there is another server running. This fixes #3501. #12133 (alexey-milovidov).
- Fixed handling dependency of table with `ENGINE=Dictionary` on dictionary. This fixes #10994. This fixes #10397. #12116 (Vitaly Baranov).
- Fixed performance for selects with `UNION` caused by wrong limit for the total number of threads. Fixes #12030. #12103 (Nikolai Kochetov).
- Fixed segfault with `-StateResample` combinators. #12092 (Anton Popov).
- Fixed empty `result_rows` and `result_bytes` metrics in `system.quey_log` for selects. Fixes #11595. #12089 (Nikolai Kochetov).
- Fixed unnecessary limiting the number of threads for selects from `VIEW`. Fixes #11937. #12085 (Nikolai Kochetov).
- Fixed possible crash while using wrong type for `PREDWHERE`. Fixes #12053, #12060. #12060 (Nikolai Kochetov).
- Fixed error `Expected single dictionary argument for function` for function `defaultValueOfArgumentType` with `LowCardinality` type. Fixes #11808. #12056 (Nikolai Kochetov).
- Fixed error `Cannot capture column` for higher-order functions with `Tuple(LowCardinality)` argument. Fixes #9766. #12055 (Nikolai Kochetov).
- Parse tables metadata in parallel when loading database. This fixes slow server startup when there are large number of tables. #12045 (tavplubix).
- Make `topK` aggregate function return `Enum` for `Enum` types. This fixes #3740. #12043 (alexey-milovidov).
- Fixed constraints check if constraint is a constant expression. This fixes #11360. #12042 (alexey-milovidov).
- Fixed incorrect comparison of tuples with `Nullable` columns. Fixes #11985. #12039 (Nikolai Kochetov).
- Fixed calculation of access rights when `allow_introspection_functions=0`. #12031 (Vitaly Baranov).
- Fixed wrong result and potential crash when invoking function `if` with arguments of type `FixedString` with different sizes. This fixes #11362. #12021 (alexey-milovidov).
- A query with function `neighbor` as the only returned expression may return empty result if the function is called with offset -9223372036854775808. This fixes #11367. #12019 (alexey-milovidov).
- Fixed calculation of access rights when `allow_ddl=0`. #12015 (Vitaly Baranov).

- Fixed potential array size overflow in generateRandom that may lead to crash. This fixes #11371. #12013 (alexey-milovidov).
- Fixed potential floating point exception. This closes #11378. #12005 (alexey-milovidov).
- Fixed wrong setting name in log message at server startup. #11997 (alexey-milovidov).
- Fixed Query parameter was not set in Values format. Fixes #11918. #11936 (tavplubix).
- Keep aliases for substitutions in query (parametrized queries). This fixes #11914. #11916 (alexey-milovidov).
- Fixed bug with no moves when changing storage policy from default one. #11893 (Vladimir Chebotarev).
- Fixed potential floating point exception when parsing `DateTime64`. This fixes #11374. #11875 (alexey-milovidov).
- Fixed memory accounting via HTTP interface (can be significant with `wait_end_of_query=1`). #11840 (Azat Khuzhin).
- Parse metadata stored in zookeeper before checking for equality. #11739 (Azat Khuzhin).

## Performance Improvement

- Index not used for IN operator with literals, performance regression introduced around v19.3. This fixes #10574. #12062 (nvartolomei).

## Build/Testing/Packaging Improvement

- Install `ca-certificates` before the first `apt-get update` in Dockerfile. #12095 (Ivan Blinkov).

# ClickHouse release v20.4.6.53-stable 2020-06-25

## Bug Fix

- Fix rare crash caused by using `Nullable` column in prewhere condition. Continuation of #11608. #11869 (Nikolai Kochetov).
- Don't allow arrayJoin inside higher order functions. It was leading to broken protocol synchronization. This closes #3933. #11846 (alexey-milovidov).
- Fix wrong result of comparison of `FixedString` with constant String. This fixes #11393. This bug appeared in version 20.4. #11828 (alexey-milovidov).
- Fix wrong result for `if()` with NULLs in condition. #11807 (Artem Zuikov).
- Fix using too many threads for queries. #11788 (Nikolai Kochetov).
- Fix unexpected behaviour of queries like `SELECT *, xyz.*` which were success while an error expected. #11753 (hexiaoting).
- Now replicated fetches will be cancelled during metadata alter. #11744 (alesapin).
- Fixed `LOGICAL_ERROR` caused by wrong type deduction of complex literals in `Values` input format. #11732 (tavplubix).
- Fix `ORDER BY ... WITH FILL` over const columns. #11697 (Anton Popov).
- Pass proper timeouts when communicating with XDBC bridge. Recently timeouts were not respected when checking bridge liveness and receiving meta info. #11690 (alexey-milovidov).

- Fix `LIMIT n WITH TIES` usage together with `ORDER BY` statement, which contains aliases. #11689 (Anton Popov).
- Fix error which leads to an incorrect state of `system.mutations`. It may show that whole mutation is already done but the server still has `MUTATE_PART` tasks in the replication queue and tries to execute them. This fixes #11611. #11681 (alesapin).
- Add support for regular expressions with case-insensitive flags. This fixes #11101 and fixes #11506. #11649 (alexey-milovidov).
- Remove trivial count query optimization if row-level security is set. In previous versions the user get total count of records in a table instead filtered. This fixes #11352. #11644 (alexey-milovidov).
- Fix bloom filters for String (data skipping indices). #11638 (Azat Khuzhin).
- Fix rare crash caused by using `Nullable` column in prewhere condition. (Probably it is connected with #11572 somehow). #11608 (Nikolai Kochetov).
- Fix error `Block structure mismatch` for queries with sampling reading from `Buffer` table. #11602 (Nikolai Kochetov).
- Fix wrong exit code of the clickhouse-client, when `exception.code() % 256 = 0`. #11601 (filimonov).
- Fix trivial error in log message about "Mark cache size was lowered" at server startup. This closes #11399. #11589 (alexey-milovidov).
- Fix error `Size of offsets does not match size of column` for queries with `PREWHERE` column in (subquery) and `ARRAY JOIN`. #11580 (Nikolai Kochetov).
- Fixed rare segfault in `SHOW CREATE TABLE` Fixes #11490. #11579 (tavplubix).
- All queries in HTTP session have had the same `query_id`. It is fixed. #11578 (tavplubix).
- Now clickhouse-server docker container will prefer IPv6 checking server aliveness. #11550 (Ivan Starkov).
- Fix shard\_num/replica\_num for `<node>` (breaks `use_compact_format_in_distributed_parts_names`). #11528 (Azat Khuzhin).
- Fix race condition which may lead to an exception during table drop. It's a bit tricky and not dangerous at all. If you want an explanation, just notice me in telegram. #11523 (alesapin).
- Fix memory leak when exception is thrown in the middle of aggregation with -State functions. This fixes #8995. #11496 (alexey-milovidov).
- If data skipping index is dependent on columns that are going to be modified during background merge (for SummingMergeTree, AggregatingMergeTree as well as for TTL GROUP BY), it was calculated incorrectly. This issue is fixed by moving index calculation after merge so the index is calculated on merged data. #11162 (Azat Khuzhin).
- Get rid of old libunwind patches. <https://github.com/ClickHouse-Extras/libunwind/commit/500aa227911bd185a94bfc071d68f4d3b03cb3b1#r39048012> This allows to disable `-fno-omit-frame-pointer` in clang builds that improves performance at least by 1% in average. #10761 (Amos Bird).
- Fix usage of primary key wrapped into a function with 'FINAL' modifier and 'ORDER BY' optimization. #10715 (Anton Popov).

## Build/Testing/Packaging Improvement

- Fix several non significant errors in unit tests. #11262 (alesapin).

- Fix (false) MSan report in MergeTreeIndexFullText. The issue first appeared in #9968. #10801 (alexey-milovidov).

## ClickHouse release v20.4.5.36-stable 2020-06-10

### Bug Fix

- Fix the error Data compressed with different methods that can happen if min\_bytes\_to\_use\_direct\_io is enabled and PREWHERE is active and using SAMPLE or high number of threads. This fixes #11539. #11540 (alexey-milovidov).
- Fix return compressed size for codecs. #11448 (Nikolai Kochetov).
- Fix server crash when a column has compression codec with non-literal arguments. Fixes #11365. #11431 (alesapin).
- Fix pointInPolygon with nan as point. Fixes #11375. #11421 (Alexey Ilyukhov).
- Fix potential uninitialized memory read in MergeTree shutdown if table was not created successfully. #11420 (alexey-milovidov).
- Fixed geohashesInBox with arguments outside of latitude/longitude range. #11403 (Vasily Nemkov).
- Fix possible Pipeline stuck error for queries with external sort and limit. Fixes #11359. #11366 (Nikolai Kochetov).
- Remove redundant lock during parts send in ReplicatedMergeTree. #11354 (alesapin).
- Fix support for \G (vertical output) in clickhouse-client in multiline mode. This closes #9933. #11350 (alexey-milovidov).
- Fix potential segfault when using Lazy database. #11348 (alexey-milovidov).
- Fix crash in quantilesExactWeightedArray. #11337 (Nikolai Kochetov).
- Now merges stopped before change metadata in ALTER queries. #11335 (alesapin).
- Make writing to MATERIALIZED VIEW with setting parallel\_view\_processing = 1 parallel again. Fixes #10241. #11330 (Nikolai Kochetov).
- Fix visitParamExtractRaw when extracted JSON has strings with unbalanced { or [. #11318 (Ewout).
- Fix very rare race condition in ThreadPool. #11314 (alexey-milovidov).
- Fix insignificant data race in clickhouse-copier. Found by integration tests. #11313 (alexey-milovidov).
- Fix potential uninitialized memory in conversion. Example: SELECT toIntervalSecond(now64()). #11311 (alexey-milovidov).
- Fix the issue when index analysis cannot work if a table has Array column in primary key and if a query is filtering by this column with empty or notEmpty functions. This fixes #11286. #11303 (alexey-milovidov).
- Fix bug when query speed estimation can be incorrect and the limit of min\_execution\_speed may not work or work incorrectly if the query is throttled by max\_network\_bandwidth, max\_execution\_speed or priority settings. Change the default value of timeout\_before\_checking\_execution\_speed to non-zero, because otherwise the settings min\_execution\_speed and max\_execution\_speed have no effect. This fixes #11297. This fixes #5732. This fixes #6228. Usability improvement: avoid concatenation of exception message with progress bar in clickhouse-client. #11296 (alexey-milovidov).

- Fix crash when SET DEFAULT ROLE is called with wrong arguments. This fixes #10586. #11278 (Vitaly Baranov).
- Fix crash while reading malformed data in Protobuf format. This fixes #5957, fixes #11203. #11258 (Vitaly Baranov).
- Fixed a bug when cache-dictionary could return default value instead of normal (when there are only expired keys). This affects only string fields. #11233 (Nikita Mikhaylov).
- Fix error `Block structure mismatch` in `QueryPipeline` while reading from `VIEW` with constants in inner query. Fixes #11181. #11205 (Nikolai Kochetov).
- Fix possible exception `Invalid status for associated output` #11200 (Nikolai Kochetov).
- Fix possible error `Cannot capture column` for higher-order functions with `Array(Array(LowCardinality))` captured argument. #11185 (Nikolai Kochetov).
- Fixed S3 globbing which could fail in case of more than 1000 keys and some backends. #11179 (Vladimir Chebotarev).
- If data skipping index is dependent on columns that are going to be modified during background merge (for SummingMergeTree, AggregatingMergeTree as well as for TTL GROUP BY), it was calculated incorrectly. This issue is fixed by moving index calculation after merge so the index is calculated on merged data. #11162 (Azat Khuzhin).
- Fix Kafka performance issue related to reschedules based on limits, which were always applied. #11149 (filimonov).
- Fix for the hang which was happening sometimes during DROP of table engine=Kafka (or during server restarts). #11145 (filimonov).
- Fix excessive reserving of threads for simple queries (optimization for reducing the number of threads, which was partly broken after changes in pipeline). #11114 (Azat Khuzhin).
- Fix predicates optimization for distributed queries (`enable_optimize_predicate_expression=1`) for queries with `HAVING` section (i.e. when filtering on the server initiator is required), by preserving the order of expressions (and this is enough to fix), and also force aggregator use column names over indexes. Fixes: #10613, #11413. #10621 (Azat Khuzhin).

## Build/Testing/Packaging Improvement

- Fix several flaky integration tests. #11355 (alesapin).

## ClickHouse release v20.4.4.18-stable 2020-05-26

No changes compared to v20.4.3.16-stable.

## ClickHouse release v20.4.3.16-stable 2020-05-23

### Bug Fix

- Removed logging from mutation finalization task if nothing was finalized. #11109 (alesapin).
- Fixed memory leak in `registerDiskS3`. #11074 (Pavel Kovalenko).
- Fixed the potential missed data during termination of Kafka engine table. #11048 (filimonov).
- Fixed `parseDateTime64BestEffort` argument resolution bugs. #11038 (Vasily Nemkov).
- Fixed very rare potential use-after-free error in `MergeTree` if table was not created successfully. #10986, #10970 (alexey-milovidov).

- Fixed metadata (relative path for rename) and data (relative path for symlink) handling for Atomic database. #10980 (Azat Khuzhin).
- Fixed server crash on concurrent `ALTER` and `DROP DATABASE` queries with Atomic database engine. #10968 (tavplubix).
- Fixed incorrect raw data size in `getRawData()` method. #10964 (lgr).
- Fixed incompatibility of two-level aggregation between versions 20.1 and earlier. This incompatibility happens when different versions of ClickHouse are used on initiator node and remote nodes and the size of GROUP BY result is large and aggregation is performed by a single String field. It leads to several unmerged rows for a single key in result. #10952 (alexey-milovidov).
- Fixed sending partially written files by the `DistributedBlockOutputStream`. #10940 (Azat Khuzhin).
- Fixed crash in `SELECT count(notNullIn(NULL, []))`. #10920 (Nikolai Kochetov).
- Fixed the hang which was happening sometimes during `DROP` of Kafka table engine. (or during server restarts). #10910 (filimonov).
- Fixed the impossibility of executing multiple `ALTER RENAME` like `a TO b, c TO a`. #10895 (alesapin).
- Fixed possible race which could happen when you get result from aggregate function state from multiple thread for the same column. The only way it can happen is when you use `finalizeAggregation` function while reading from table with `Memory` engine which stores `AggregateFunction` state for `quantile*` function. #10890 (Nikolai Kochetov).
- Fixed backward compatibility with tuples in Distributed tables. #10889 (Anton Popov).
- Fixed `SIGSEGV` in `StringHashTable` if such a key does not exist. #10870 (Azat Khuzhin).
- Fixed `WATCH` hangs after `LiveView` table was dropped from database with `Atomic` engine. #10859 (tavplubix).
- Fixed bug in `ReplicatedMergeTree` which might cause some `ALTER` on `OPTIMIZE` query to hang waiting for some replica after it become inactive. #10849 (tavplubix).
- Now constraints are updated if the column participating in `CONSTRAINT` expression was renamed. Fixes #10844. #10847 (alesapin).
- Fixed potential read of uninitialized memory in cache-dictionary. #10834 (alexey-milovidov).
- Fixed columns order after `Block::sortColumns()`. #10826 (Azat Khuzhin).
- Fixed the issue with `ODBC` bridge when no quoting of identifiers is requested. Fixes #7984. #10821 (alexey-milovidov).
- Fixed UBSan and MSan report in `DateLUT`. #10798 (alexey-milovidov).
- Fixed incorrect type conversion in key conditions. Fixes #6287. #10791 (Andrew Onyshchuk).
- Fixed parallel\_view\_processing behavior. Now all insertions into `MATERIALIZED VIEW` without exception should be finished if exception happened. Fixes #10241. #10757 (Nikolai Kochetov).
- Fixed combinator `-OrNull` and `-OrDefault` when combined with `-State`. #10741 (hcz).
- Fixed possible buffer overflow in function `h3EdgeAngle`. #10711 (alexey-milovidov).
- Fixed bug which locks concurrent alters when table has a lot of parts. #10659 (alesapin).
- Fixed nullptr dereference in `StorageBuffer` if server was shutdown before table startup. #10641 (alexey-milovidov).

- Fixed `optimize_skip_unused_shards` with `LowCardinality`. #10611 (Azat Khuzhin).
- Fixed handling condition variable for synchronous mutations. In some cases signals to that condition variable could be lost. #10588 (Vladimir Chebotarev).
- Fixed possible crash when `createDictionary()` is called before `loadStoredObject()` has finished. #10587 (Vitaly Baranov).
- Fixed `SELECT` of column ALIAS which default expression type different from column type. #10563 (Azat Khuzhin).
- Implemented comparison between `DateTime64` and String values. #10560 (Vasily Nemkov).
- Disable `GROUP BY sharding_key` optimization by default (`optimize_distributed_group_by_sharding_key` had been introduced and turned off by default, due to trickery of `sharding_key` analyzing, simple example is `if` in sharding key) and fix it for `WITH ROLLUP/CUBE/TOTALS`. #10516 (Azat Khuzhin).
- Fixed #10263. #10486 (Azat Khuzhin).
- Added tests about `max_rows_to_sort` setting. #10268 (alexey-milovidov).
- Added backward compatibility for create bloom filter index. #10551. #10569 (Winter Zhang).

## 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 #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 dowloading from other replicas. #10379 (alesapin)
- Add function `JSONExtractKeysAndValuesRaw` which extracts raw data from JSON objects #10378 (hcz)
- 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_wait_timeout_milliseconds`, which is one minute by default. The first settings 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 (hcz)
- 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](#))
- 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/replica_name/metadata` node does not 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 does not 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 (hczi)
- 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 does not 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 does not 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)
- New 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 (hcz)
- 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)
- 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. #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 v20.3.21.2-lts, 2020-11-02

#### Bug Fix

- Fix dictGet in sharding\_key (and similar places, i.e. when the function context is stored permanently). #16205 (Azat Khuzhin).
- Fix incorrect empty result for query from Distributed table if query has WHERE, PREWHERE and GLOBAL IN. Fixes #15792. #15933 (Nikolai Kochetov).
- Fix missing or excessive headers in TSV/CSVWithNames formats. This fixes #12504. #13343 (Azat Khuzhin).

### ClickHouse release v20.3.20.6-lts, 2020-10-09

#### Bug Fix

- Mutation might hang waiting for some non-existent part after MOVE or REPLACE PARTITION or, in rare cases, after DETACH or DROP PARTITION. It's fixed. #15724, #15537 (tavplubix).

- Fix hang of queries with a lot of subqueries to same table of MySQL engine. Previously, if there were more than 16 subqueries to same MySQL table in query, it hang forever. #15299 (Anton Popov).
- Fix 'Unknown identifier' in GROUP BY when query has JOIN over Merge table. #15242 (Artem Zuikov).
- Fix to make predicate push down work when subquery contains finalizeAggregation function. Fixes #14847. #14937 (filimonov).
- Concurrent ALTER ... REPLACE/MOVE PARTITION ... queries might cause deadlock. It's fixed. #13626 (tavplubix).

## ClickHouse release v20.3.19.4-lts, 2020-09-18

### Bug Fix

- Fix rare error in SELECT queries when the queried column has DEFAULT expression which depends on the other column which also has DEFAULT and not present in select query and not exists on disk. Partially fixes #14531. #14845 (alesapin).
- Fix bug when ALTER UPDATE mutation with Nullable column in assignment expression and constant value (like UPDATE x = 42) leads to incorrect value in column or segfault. Fixes #13634, #14045. #14646 (alesapin).
- Fix wrong Decimal multiplication result caused wrong decimal scale of result column. #14603 (Artem Zuikov).

### Improvement

- Support custom codecs in compact parts. #12183 (Anton Popov).

## ClickHouse release v20.3.18.10-lts, 2020-09-08

### Bug Fix

- Stop query execution if exception happened in PipelineExecutor itself. This could prevent rare possible query hung. Continuation of #14334. #14402 (Nikolai Kochetov).
- Fixed the behaviour when sometimes cache-dictionary returned default value instead of present value from source. #13624 (Nikita Mikhaylov).
- Fix parsing row policies from users.xml when names of databases or tables contain dots. This fixes #5779, #12527. #13199 (Vitaly Baranov).
- Fix CAST(Nullable(String), Enum()). #12745 (Azat Khuzhin).
- Fixed data race in text\_log. It does not correspond to any real bug. #9726 (alexey-milovidov).

### Improvement

- Fix wrong error for long queries. It was possible to get syntax error other than Max query size exceeded for correct query. #13928 (Nikolai Kochetov).
- Return NULL/zero when value is not parsed completely in parseDateTimeBestEffortOrNull/Zero functions. This fixes #7876. #11653 (alexey-milovidov).

### Performance Improvement

- Slightly optimize very short queries with LowCardinality. #14129 (Anton Popov).

### Build/Testing/Packaging Improvement

- Fix UBSan report (adding zero to nullptr) in HashTable that appeared after migration to clang-10. #10638 (alexey-milovidov).

## ClickHouse release v20.3.17.173-lts, 2020-08-15

### Bug Fix

- Fix crash in JOIN with StorageMerge and `set enable_optimize_predicate_expression=1`. [#13679](#) ([Artem Zuikov](#)).
- Fix invalid return type for comparison of tuples with `NULL` elements. Fixes [#12461](#). [#13420](#) ([Nikolai Kochetov](#)).
- Fix queries with constant columns and `ORDER BY` prefix of primary key. [#13396](#) ([Anton Popov](#)).
- Return passed number for numbers with MSB set in `roundUpToPowerOfTwoOrZero()`. [#13234](#) ([Azat Khuzhin](#)).

## ClickHouse release v20.3.16.165-lts 2020-08-10

### Bug Fix

- Fixed error in `parseDateTimeBestEffort` function when unix timestamp was passed as an argument. This fixes [#13362](#). [#13441](#) ([alexey-milovidov](#)).
- Fixed potentially low performance and slightly incorrect result for `uniqExact`, `topK`, `sumDistinct` and similar aggregate functions called on Float types with `NaN` values. It also triggered assert in debug build. This fixes [#12491](#). [#13254](#) ([alexey-milovidov](#)).
- Fixed function if with nullable `constexpr` as cond that is not literal `NULL`. Fixes [#12463](#). [#13226](#) ([alexey-milovidov](#)).
- Fixed assert in `arrayElement` function in case of array elements are Nullable and array subscript is also Nullable. This fixes [#12172](#). [#13224](#) ([alexey-milovidov](#)).
- Fixed unnecessary limiting for the number of threads for selects from local replica. [#12840](#) ([Nikolai Kochetov](#)).
- Fixed possible extra overflow row in data which could appear for queries `WITH TOTALS`. [#12747](#) ([Nikolai Kochetov](#)).
- Fixed performance with large tuples, which are interpreted as functions in `IN` section. The case when user write `WHERE x IN tuple(1, 2, ...)` instead of `WHERE x IN (1, 2, ...)` for some obscure reason. [#12700](#) ([Anton Popov](#)).
- Fixed memory tracking for `input_format_parallel_parsing` (by attaching thread to group). [#12672](#) ([Azat Khuzhin](#)).
- Fixed [#12293](#) allow push predicate when subquery contains `with` clause. [#12663](#) ([Winter Zhang](#)).
- Fixed [#10572](#) fix bloom filter index with const expression. [#12659](#) ([Winter Zhang](#)).
- Fixed SIGSEGV in StorageKafka when broker is unavailable (and not only). [#12658](#) ([Azat Khuzhin](#)).
- Fixed race condition in external dictionaries with cache layout which can lead server crash. [#12566](#) ([alesapin](#)).
- Fixed bug which lead to broken old parts after `ALTER DELETE` query when `enable_mixed_granularity_parts=1`. Fixes [#12536](#). [#12543](#) ([alesapin](#)).
- Better exception for function `in` with invalid number of arguments. [#12529](#) ([Anton Popov](#)).
- Fixed performance issue, while reading from compact parts. [#12492](#) ([Anton Popov](#)).
- Fixed the deadlock if `text_log` is enabled. [#12452](#) ([alexey-milovidov](#)).

- Fixed possible segfault if StorageMerge. Closes #12054. #12401 (tavplubix).
- Fixed TOTALS/ROLLUP/CUBE for aggregate functions with `-State` and `Nullable` arguments. This fixes #12163. #12376 (alexey-milovidov).
- Fixed order of columns in `WITH FILL` modifier. Previously order of columns of `ORDER BY` statement wasn't respected. #12306 (Anton Popov).
- Avoid "bad cast" exception when there is an expression that filters data by virtual columns (like `_table` in `Merge` tables) or by "index" columns in system tables such as filtering by database name when querying from `system.tables`, and this expression returns `Nullable` type. This fixes #12166. #12305 (alexey-milovidov).
- Show error after `TrieDictionary` failed to load. #12290 (Vitaly Baranov).
- The function `arrayFill` worked incorrectly for empty arrays that may lead to crash. This fixes #12263. #12279 (alexey-milovidov).
- Implement conversions to the common type for `LowCardinality` types. This allows to execute UNION ALL of tables with columns of `LowCardinality` and other columns. This fixes #8212. This fixes #4342. #12275 (alexey-milovidov).
- Fixed the behaviour when during multiple sequential inserts in `StorageFile` header for some special types was written more than once. This fixed #6155. #12197 (Nikita Mikhaylov).
- Fixed logical functions for `UInt8` values when they are not equal to 0 or 1. #12196 (Alexander Kazakov).
- Fixed `dictGet` arguments check during GROUP BY injective functions elimination. #12179 (Azat Khuzhin).
- Fixed wrong logic in `ALTER DELETE` that leads to deleting of records when condition evaluates to NULL. This fixes #9088. This closes #12106. #12153 (alexey-milovidov).
- Fixed transform of query to send to external DBMS (e.g. MySQL, ODBC) in presence of aliases. This fixes #12032. #12151 (alexey-milovidov).
- Fixed potential overflow in integer division. This fixes #12119. #12140 (alexey-milovidov).
- Fixed potential infinite loop in `greatCircleDistance`, `geoDistance`. This fixes #12117. #12137 (alexey-milovidov).
- Avoid `There is no query` exception for materialized views with joins or with subqueries attached to system logs (`system.query_log`, `metric_log`, etc) or to engine=Buffer underlying table. #12120 (filimonov).
- Fixed performance for selects with `UNION` caused by wrong limit for the total number of threads. Fixes #12030. #12103 (Nikolai Kochetov).
- Fixed segfault with `-StateResample` combinators. #12092 (Anton Popov).
- Fixed unnecessary limiting the number of threads for selects from `VIEW`. Fixes #11937. #12085 (Nikolai Kochetov).
- Fixed possible crash while using wrong type for `PREWHERE`. Fixes #12053, #12060. #12060 (Nikolai Kochetov).
- Fixed error `Expected single dictionary argument for function` for function `defaultValueOfArgumentType` with `LowCardinality` type. Fixes #11808. #12056 (Nikolai Kochetov).
- Fixed error `Cannot capture column for higher-order functions with Tuple(LowCardinality) argument`. Fixes #9766. #12055 (Nikolai Kochetov).

- Parse tables metadata in parallel when loading database. This fixes slow server startup when there are large number of tables. #12045 (tavplubix).
- Make `topK` aggregate function return `Enum` for `Enum` types. This fixes #3740. #12043 (alexey-milovidov).
- Fixed constraints check if constraint is a constant expression. This fixes #11360. #12042 (alexey-milovidov).
- Fixed incorrect comparison of tuples with `Nullable` columns. Fixes #11985. #12039 (Nikolai Kochetov).
- Fixed wrong result and potential crash when invoking function `if` with arguments of type `FixedString` with different sizes. This fixes #11362. #12021 (alexey-milovidov).
- A query with function `neighbor` as the only returned expression may return empty result if the function is called with offset -9223372036854775808. This fixes #11367. #12019 (alexey-milovidov).
- Fixed potential array size overflow in `generateRandom` that may lead to crash. This fixes #11371. #12013 (alexey-milovidov).
- Fixed potential floating point exception. This closes #11378. #12005 (alexey-milovidov).
- Fixed wrong setting name in log message at server startup. #11997 (alexey-milovidov).
- Fixed Query parameter was not set in `Values` format. Fixes #11918. #11936 (tavplubix).
- Keep aliases for substitutions in query (parametrized queries). This fixes #11914. #11916 (alexey-milovidov).
- Fixed potential floating point exception when parsing `DateTime64`. This fixes #11374. #11875 (alexey-milovidov).
- Fixed memory accounting via `HTTP` interface (can be significant with `wait_end_of_query=1`). #11840 (Azat Khuzhin).
- Fixed wrong result for `if()` with `NULLs` in condition. #11807 (Artem Zuikov).
- Parse metadata stored in zookeeper before checking for equality. #11739 (Azat Khuzhin).
- Fixed `LIMIT n WITH TIES` usage together with `ORDER BY` statement, which contains aliases. #11689 (Anton Popov).
- Fix potential read of uninitialized memory in cache dictionary. #10834 (alexey-milovidov).

## Performance Improvement

- Index not used for `IN` operator with literals, performance regression introduced around v19.3. This fixes #10574. #12062 (nvartolomei).

## ClickHouse release v20.3.12.112-lts 2020-06-25

### Bug Fix

- Fix rare crash caused by using `Nullable` column in prewhere condition. Continuation of #11608. #11869 (Nikolai Kochetov).
- Don't allow `arrayJoin` inside higher order functions. It was leading to broken protocol synchronization. This closes #3933. #11846 (alexey-milovidov).
- Fix using too many threads for queries. #11788 (Nikolai Kochetov).

- Fix unexpected behaviour of queries like `SELECT *, xyz.*` which were success while an error expected. [#11753 \(hexiaoting\)](#).
- Now replicated fetches will be cancelled during metadata alter. [#11744 \(alesapin\)](#).
- Fixed `LOGICAL_ERROR` caused by wrong type deduction of complex literals in `Values` input format. [#11732 \(tavplubix\)](#).
- Fix `ORDER BY ... WITH FILL` over const columns. [#11697 \(Anton Popov\)](#).
- Pass proper timeouts when communicating with XDBC bridge. Recently timeouts were not respected when checking bridge liveness and receiving meta info. [#11690 \(alexey-milovidov\)](#).
- Fix error which leads to an incorrect state of `system.mutations`. It may show that whole mutation is already done but the server still has `MUTATE_PART` tasks in the replication queue and tries to execute them. This fixes [#11611](#). [#11681 \(alesapin\)](#).
- Add support for regular expressions with case-insensitive flags. This fixes [#11101](#) and fixes [#11506](#). [#11649 \(alexey-milovidov\)](#).
- Remove trivial count query optimization if row-level security is set. In previous versions the user get total count of records in a table instead filtered. This fixes [#11352](#). [#11644 \(alexey-milovidov\)](#).
- Fix bloom filters for String (data skipping indices). [#11638 \(Azat Khuzhin\)](#).
- Fix rare crash caused by using `Nullable` column in prewhere condition. (Probably it is connected with [#11572](#) somehow). [#11608 \(Nikolai Kochetov\)](#).
- Fix error `Block structure mismatch` for queries with sampling reading from `Buffer` table. [#11602 \(Nikolai Kochetov\)](#).
- Fix wrong exit code of the clickhouse-client, when `exception.code() % 256 = 0`. [#11601 \(filimonov\)](#).
- Fix trivial error in log message about "Mark cache size was lowered" at server startup. This closes [#11399](#). [#11589 \(alexey-milovidov\)](#).
- Fix error `Size of offsets does not match size of column` for queries with `PREWHERE` column in (subquery) and `ARRAY JOIN`. [#11580 \(Nikolai Kochetov\)](#).
- All queries in HTTP session have had the same `query_id`. It is fixed. [#11578 \(tavplubix\)](#).
- Now clickhouse-server docker container will prefer IPv6 checking server aliveness. [#11550 \(Ivan Starkov\)](#).
- Fix `shard_num/replica_num` for `<node>` (breaks `use_compact_format_in_distributed_parts_names`). [#11528 \(Azat Khuzhin\)](#).
- Fix memory leak when exception is thrown in the middle of aggregation with -State functions. This fixes [#8995](#). [#11496 \(alexey-milovidov\)](#).
- Fix wrong results of distributed queries when alias could override qualified column name. Fixes [#9672](#) [#9714](#). [#9972 \(Artem Zuikov\)](#).

## ClickHouse release v20.3.11.97-Its 2020-06-10

### New Feature

- 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_expired_keys` settings (to forbid reading very expired keys). [#10337 \(Nikita Mikhaylov\)](#).

## Bug Fix

- Fix the error Data compressed with different methods that can happen if `min_bytes_to_use_direct_io` is enabled and PREWHERE is active and using SAMPLE or high number of threads. This fixes #11539. #11540 (alexey-milovidov).
- Fix return compressed size for codecs. #11448 (Nikolai Kochetov).
- Fix server crash when a column has compression codec with non-literal arguments. Fixes #11365. #11431 (alesapin).
- Fix pointInPolygon with nan as point. Fixes #11375. #11421 (Alexey Ilyukhov).
- Fix crash in JOIN over LowCarinality(T) and Nullable(T). #11380. #11414 (Artem Zuikov).
- Fix error code for wrong USING key. #11373. #11404 (Artem Zuikov).
- Fixed geohashesInBox with arguments outside of latitude/longitude range. #11403 (Vasily Nemkov).
- Better errors for `joinGet()` functions. #11389 (Artem Zuikov).
- Fix possible Pipeline stuck error for queries with external sort and limit. Fixes #11359. #11366 (Nikolai Kochetov).
- Remove redundant lock during parts send in ReplicatedMergeTree. #11354 (alesapin).
- Fix support for \G (vertical output) in clickhouse-client in multiline mode. This closes #9933. #11350 (alexey-milovidov).
- Fix crash in direct selects from StorageJoin (without JOIN) and wrong nullability. #11340 (Artem Zuikov).
- Fix crash in `quantilesExactWeightedArray`. #11337 (Nikolai Kochetov).
- Now merges stopped before change metadata in ALTER queries. #11335 (alesapin).
- Make writing to MATERIALIZED VIEW with setting `parallel_view_processing = 1` parallel again. Fixes #10241. #11330 (Nikolai Kochetov).
- Fix visitParamExtractRaw when extracted JSON has strings with unbalanced { or [. #11318 (Ewout).
- Fix very rare race condition in ThreadPool. #11314 (alexey-milovidov).
- Fix potential uninitialized memory in conversion. Example: `SELECT toIntervalSecond(now64())`. #11311 (alexey-milovidov).
- Fix the issue when index analysis cannot work if a table has Array column in primary key and if a query is filtering by this column with `empty` or `notEmpty` functions. This fixes #11286. #11303 (alexey-milovidov).
- Fix bug when query speed estimation can be incorrect and the limit of `min_execution_speed` may not work or work incorrectly if the query is throttled by `max_network_bandwidth`, `max_execution_speed` or `priority` settings. Change the default value of `timeout_before_checking_execution_speed` to non-zero, because otherwise the settings `min_execution_speed` and `max_execution_speed` have no effect. This fixes #11297. This fixes #5732. This fixes #6228. Usability improvement: avoid concatenation of exception message with progress bar in clickhouse-client. #11296 (alexey-milovidov).
- Fix crash while reading malformed data in Protobuf format. This fixes #5957, fixes #11203. #11258 (Vitaly Baranov).
- Fixed a bug when cache-dictionary could return default value instead of normal (when there are only expired keys). This affects only string fields. #11233 (Nikita Mikhaylov).

- Fix error `Block structure mismatch` in `QueryPipeline` while reading from `VIEW` with constants in inner query. Fixes [#11181](#). [#11205](#) ([Nikolai Kochetov](#)).
- Fix possible exception `Invalid status for associated output` [#11200](#) ([Nikolai Kochetov](#)).
- Fix possible error `Cannot capture column` for higher-order functions with `Array(Array(LowCardinality))` captured argument. [#11185](#) ([Nikolai Kochetov](#)).
- Fixed S3 globbing which could fail in case of more than 1000 keys and some backends. [#11179](#) ([Vladimir Chebotarev](#)).
- If data skipping index is dependent on columns that are going to be modified during background merge (for `SummingMergeTree`, `AggregatingMergeTree` as well as for TTL GROUP BY), it was calculated incorrectly. This issue is fixed by moving index calculation after merge so the index is calculated on merged data. [#11162](#) ([Azat Khuzhin](#)).
- Fix excessive reserving of threads for simple queries (optimization for reducing the number of threads, which was partly broken after changes in pipeline). [#11114](#) ([Azat Khuzhin](#)).
- Fix predicates optimization for distributed queries (`enable_optimize_predicate_expression=1`) for queries with `HAVING` section (i.e. when filtering on the server initiator is required), by preserving the order of expressions (and this is enough to fix), and also force aggregator use column names over indexes. Fixes: [#10613](#), [#11413](#). [#10621](#) ([Azat Khuzhin](#)).
- Introduce commit retry logic to decrease the possibility of getting duplicates from Kafka in rare cases when offset commit was failed. [#9884](#) ([filimonov](#)).

## Performance Improvement

- Get dictionary and check access rights only once per each call of any function reading external dictionaries. [#10928](#) ([Vitaly Baranov](#)).

## Build/Testing/Packaging Improvement

- Fix several flaky integration tests. [#11355](#) ([alesapin](#)).

# ClickHouse release v20.3.10.75-Its 2020-05-23

## Bug Fix

- Removed logging from mutation finalization task if nothing was finalized. [#11109](#) ([alesapin](#)).
- Fixed `parseDateTime64BestEffort` argument resolution bugs. [#11038](#) ([Vasily Nemkov](#)).
- Fixed incorrect raw data size in method `getRawData()`. [#10964](#) ([lgr](#)).
- Fixed incompatibility of two-level aggregation between versions 20.1 and earlier. This incompatibility happens when different versions of ClickHouse are used on initiator node and remote nodes and the size of `GROUP BY` result is large and aggregation is performed by a single `String` field. It leads to several unmerged rows for a single key in result. [#10952](#) ([alexey-milovidov](#)).
- Fixed backward compatibility with tuples in `Distributed` tables. [#10889](#) ([Anton Popov](#)).
- Fixed `SIGSEGV` in `StringHashTable` if such a key does not exist. [#10870](#) ([Azat Khuzhin](#)).
- Fixed bug in `ReplicatedMergeTree` which might cause some `ALTER` on `OPTIMIZE` query to hang waiting for some replica after it become inactive. [#10849](#) ([tavplubix](#)).
- Fixed columns order after `Block::sortColumns()`. [#10826](#) ([Azat Khuzhin](#)).
- Fixed the issue with ODBC bridge when no quoting of identifiers is requested. Fixes [#7984](#). [#10821](#) ([alexey-milovidov](#)).

- Fixed UBSan and MSan report in DateLUT. #10798 (alexey-milovidov).
- Fixed incorrect type conversion in key conditions. Fixes #6287. #10791 (Andrew Onyshchuk)
- Fixed parallel\_view\_processing behavior. Now all insertions into MATERIALIZED VIEW without exception should be finished if exception happened. Fixes #10241. #10757 (Nikolai Kochetov).
- Fixed combinator -OrNull and -OrDefault when combined with -State. #10741 (hczi).
- Fixed crash in generateRandom with nested types. Fixes #10583. #10734 (Nikolai Kochetov).
- Fixed data corruption for LowCardinality(FixedString) key column in SummingMergeTree which could have happened after merge. Fixes #10489. #10721 (Nikolai Kochetov).
- Fixed possible buffer overflow in function h3EdgeAngle. #10711 (alexey-milovidov).
- Fixed disappearing totals. Totals could have been filtered if query had had join or subquery with external where condition. Fixes #10674. #10698 (Nikolai Kochetov).
- Fixed multiple usages of IN operator with the identical set in one query. #10686 (Anton Popov).
- Fixed bug, which causes http requests stuck on client close when readonly=2 and cancel\_http\_READONLY\_queries\_on\_client\_close=1. Fixes #7939, #7019, #7736, #7091. #10684 (tavplubix).
- Fixed order of parameters in AggregateTransform constructor. #10667 (palasonic1).
- Fixed the lack of parallel execution of remote queries with distributed\_aggregation\_memory\_efficient enabled. Fixes #10655. #10664 (Nikolai Kochetov).
- Fixed possible incorrect number of rows for queries with LIMIT. Fixes #10566, #10709. #10660 (Nikolai Kochetov).
- Fixed a bug which locks concurrent alters when table has a lot of parts. #10659 (alesapin).
- Fixed a bug when on SYSTEM DROP DNS CACHE query also drop caches, which are used to check if user is allowed to connect from some IP addresses. #10608 (tavplubix).
- Fixed incorrect scalar results inside inner query of MATERIALIZED VIEW in case if this query contained dependent table. #10603 (Nikolai Kochetov).
- Fixed SELECT of column ALIAS which default expression type different from column type. #10563 (Azat Khuzhin).
- Implemented comparison between DateTime64 and String values. #10560 (Vasily Nemkov).
- Fixed index corruption, which may occur in some cases after merge compact parts into another compact part. #10531 (Anton Popov).
- Fixed the situation, when mutation finished all parts, but hung up in is\_done=0. #10526 (alesapin).
- Fixed overflow at beginning of unix epoch for timezones with fractional offset from UTC. This fixes #9335. #10513 (alexey-milovidov).
- Fixed improper shutdown of Distributed storage. #10491 (Azat Khuzhin).
- Fixed numeric overflow in simpleLinearRegression over large integers. #10474 (hczi).

## Build/Testing/Packaging Improvement

- Fix UBSan report in LZ4 library. #10631 (alexey-milovidov).
- Fix clang-10 build. #10238. #10370 (Amos Bird).

- Added failing tests about `max_rows_to_sort` setting. #10268 (alexey-milovidov).
- Added some improvements in printing diagnostic info in input formats. Fixes #10204. #10418 (tavplubix).
- Added CA certificates to clickhouse-server docker image. #10476 (filimonov).

## Bug fix

- Fix error `the BloomFilter false positive must be a double number between 0 and 1` #10551. #10569 (Winter Zhang).

## 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 leaded 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 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 does not 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 does not 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).
- 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 does not 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 does not change type of column and `MODIFY` type does not 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 `system.dictionaries` table #9394 ([Nikita Mikhaylov](#))
- Add `blockSerializedSize` function (size on disk without compression) #8952 ([Azat Khuzhin](#))
- Add function `moduloOrZero` #9358 ([hc2](#))
- 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 did not 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. #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 `FixedString` 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 `LLVMEExecutableFunction` 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 does not 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` by minus one constant #9351 (hcz)
- 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 times 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 `gsasl` libraries on Mac OS X. #9000 (Andrew Onyshchuk)
- build fix on GCC 9.2.0 #9306 (vxider)

## ClickHouse release v20.1

### ClickHouse release v20.1.16.120-stable 2020-60-26

#### Bug Fix

- Fix rare crash caused by using `Nullable` column in prewhere condition. Continuation of #11608. #11869 (Nikolai Kochetov).
- Don't allow arrayJoin inside higher order functions. It was leading to broken protocol synchronization. This closes #3933. #11846 (alexey-milovidov).
- Fix unexpected behaviour of queries like `SELECT *, xyz.*` which were success while an error expected. #11753 (hexiaoting).
- Fixed `LOGICAL_ERROR` caused by wrong type deduction of complex literals in Values input format. #11732 (tavplubix).
- Fix `ORDER BY ... WITH FILL` over const columns. #11697 (Anton Popov).
- Pass proper timeouts when communicating with XDBC bridge. Recently timeouts were not respected when checking bridge liveness and receiving meta info. #11690 (alexey-milovidov).
- Add support for regular expressions with case-insensitive flags. This fixes #11101 and fixes #11506. #11649 (alexey-milovidov).
- Fix bloom filters for String (data skipping indices). #11638 (Azat Khuzhin).
- Fix rare crash caused by using `Nullable` column in prewhere condition. (Probably it is connected with #11572 somehow). #11608 (Nikolai Kochetov).
- Fix wrong exit code of the clickhouse-client, when `exception.code() % 256 = 0`. #11601 (filimonov).
- Fix trivial error in log message about "Mark cache size was lowered" at server startup. This closes #11399. #11589 (alexey-milovidov).
- Now clickhouse-server docker container will prefer IPv6 checking server aliveness. #11550 (Ivan Starkov).
- Fix memory leak when exception is thrown in the middle of aggregation with -State functions. This fixes #8995. #11496 (alexey-milovidov).
- Fix usage of primary key wrapped into a function with 'FINAL' modifier and 'ORDER BY' optimization. #10715 (Anton Popov).

### ClickHouse release v20.1.15.109-stable 2020-06-19

#### Bug Fix

- Fix excess lock for structure during alter. #11790 (alesapin).

### ClickHouse release v20.1.14.107-stable 2020-06-11

#### Bug Fix

- Fix error Size of offsets does not match size of column for queries with `PREWHERE` column in (subquery) and `ARRAY JOIN`. #11580 (Nikolai Kochetov).

# ClickHouse release v20.1.13.105-stable 2020-06-10

## Bug Fix

- Fix the error Data compressed with different methods that can happen if `min_bytes_to_use_direct_io` is enabled and PREWHERE is active and using SAMPLE or high number of threads. This fixes #11539. #11540 (alexey-milovidov).
- Fix return compressed size for codecs. #11448 (Nikolai Kochetov).
- Fix server crash when a column has compression codec with non-literal arguments. Fixes #11365. #11431 (alesapin).
- Fix pointInPolygon with nan as point. Fixes #11375. #11421 (Alexey Ilyukhov).
- Fixed geohashesInBox with arguments outside of latitude/longitude range. #11403 (Vasily Nemkov).
- Fix possible Pipeline stuck error for queries with external sort and limit. Fixes #11359. #11366 (Nikolai Kochetov).
- Fix crash in `quantilesExactWeightedArray`. #11337 (Nikolai Kochetov).
- Make writing to MATERIALIZED VIEW with setting `parallel_view_processing = 1` parallel again. Fixes #10241. #11330 (Nikolai Kochetov).
- Fix visitParamExtractRaw when extracted JSON has strings with unbalanced { or [. #11318 (Ewout).
- Fix very rare race condition in ThreadPool. #11314 (alexey-milovidov).
- Fix potential uninitialized memory in conversion. Example: `SELECT toIntervalSecond(now64())`. #11311 (alexey-milovidov).
- Fix the issue when index analysis cannot work if a table has Array column in primary key and if a query is filtering by this column with `empty` or `notEmpty` functions. This fixes #11286. #11303 (alexey-milovidov).
- Fix bug when query speed estimation can be incorrect and the limit of `min_execution_speed` may not work or work incorrectly if the query is throttled by `max_network_bandwidth`, `max_execution_speed` or `priority` settings. Change the default value of `timeout_before_checking_execution_speed` to non-zero, because otherwise the settings `min_execution_speed` and `max_execution_speed` have no effect. This fixes #11297. This fixes #5732. This fixes #6228. Usability improvement: avoid concatenation of exception message with progress bar in `clickhouse-client`. #11296 (alexey-milovidov).
- Fix crash while reading malformed data in Protobuf format. This fixes #5957, fixes #11203. #11258 (Vitaly Baranov).
- Fix possible error `Cannot capture column` for higher-order functions with `Array(Array(LowCardinality))` captured argument. #11185 (Nikolai Kochetov).
- If data skipping index is dependent on columns that are going to be modified during background merge (for SummingMergeTree, AggregatingMergeTree as well as for TTL GROUP BY), it was calculated incorrectly. This issue is fixed by moving index calculation after merge so the index is calculated on merged data. #11162 (Azat Khuzhin).
- Remove logging from mutation finalization task if nothing was finalized. #11109 (alesapin).
- Fixed `parseDateTime64BestEffort` argument resolution bugs. #10925. #11038 (Vasily Nemkov).
- Fix incorrect raw data size in method `getRawData()`. #10964 (Igr).
- Fix backward compatibility with tuples in Distributed tables. #10889 (Anton Popov).

- Fix SIGSEGV in StringHashTable (if such key does not exist). [#10870](#) ([Azat Khuzhin](#)).
- Fixed bug in ReplicatedMergeTree which might cause some ALTER on OPTIMIZE query to hang waiting for some replica after it become inactive. [#10849](#) ([tavplubix](#)).
- Fix columns order after Block::sortColumns() (also add a test that shows that it affects some real use case - Buffer engine). [#10826](#) ([Azat Khuzhin](#)).
- Fix the issue with ODBC bridge when no quoting of identifiers is requested. This fixes [#7984](#). [#10821](#) ([alexey-milovidov](#)).
- Fix UBSan and MSan report in DateLUT. [#10798](#) ([alexey-milovidov](#)).
- ▪ Make use of `src_type` for correct type conversion in key conditions. Fixes [#6287](#). [#10791](#) ([Andrew Onyshchuk](#)).
- Fix parallel\_view\_processing behavior. Now all insertions into MATERIALIZED VIEW without exception should be finished if exception happened. Fixes [#10241](#). [#10757](#) ([Nikolai Kochetov](#)).
- Fix combinator -OrNull and -OrDefault when combined with -State. [#10741](#) ([hcz](#)).
- Fix disappearing totals. Totals could have been filtered if query had had join or subquery with external where condition. Fixes [#10674](#). [#10698](#) ([Nikolai Kochetov](#)).
- Fix multiple usages of IN operator with the identical set in one query. [#10686](#) ([Anton Popov](#)).
- Fix order of parameters in AggregateTransform constructor. [#10667](#) ([palasonic1](#)).
- Fix the lack of parallel execution of remote queries with `distributed_aggregation_memory_efficient` enabled. Fixes [#10655](#). [#10664](#) ([Nikolai Kochetov](#)).
- Fix predicates optimization for distributed queries (`enable_optimize_predicate_expression=1`) for queries with HAVING section (i.e. when filtering on the server initiator is required), by preserving the order of expressions (and this is enough to fix), and also force aggregator use column names over indexes. Fixes: [#10613](#), [#11413](#). [#10621](#) ([Azat Khuzhin](#)).
- Fix error the BloomFilter false positive must be a double number between 0 and 1 [#10551](#). [#10569](#) ([Winter Zhang](#)).
- Fix SELECT of column ALIAS which default expression type different from column type. [#10563](#) ([Azat Khuzhin](#)).
- ▪ Implemented comparison between DateTime64 and String values (just like for DateTime). [#10560](#) ([Vasily Nemkov](#)).

## ClickHouse release v20.1.12.86, 2020-05-26

### Bug Fix

- Fixed incompatibility of two-level aggregation between versions 20.1 and earlier. This incompatibility happens when different versions of ClickHouse are used on initiator node and remote nodes and the size of GROUP BY result is large and aggregation is performed by a single String field. It leads to several unmerged rows for a single key in result. [#10952](#) ([alexey-milovidov](#)).
- Fixed data corruption for LowCardinality(FixedString) key column in SummingMergeTree which could have happened after merge. Fixes [#10489](#). [#10721](#) ([Nikolai Kochetov](#)).
- Fixed bug, which causes http requests stuck on client close when `readonly=2` and `cancel_http_READONLY_queries_on_client_close=1`. Fixes [#7939](#), [#7019](#), [#7736](#), [#7091](#). [#10684](#) ([tavplubix](#)).

- Fixed a bug when on `SYSTEM DROP DNS CACHE` query also drop caches, which are used to check if user is allowed to connect from some IP addresses. [#10608 \(tavplubix\)](#).
- Fixed incorrect scalar results inside inner query of `MATERIALIZED VIEW` in case if this query contained dependent table. [#10603 \(Nikolai Kochetov\)](#).
- Fixed the situation when mutation finished all parts, but hung up in `is_done=0`. [#10526 \(alesapin\)](#).
- Fixed overflow at beginning of unix epoch for timezones with fractional offset from UTC. This fixes [#9335](#). [#10513 \(alexey-milovidov\)](#).
- Fixed improper shutdown of Distributed storage. [#10491 \(Azat Khuzhin\)](#).
- Fixed numeric overflow in `simpleLinearRegression` over large integers. [#10474 \(hcz\)](#).
- Fixed removing metadata directory when attach database fails. [#10442 \(Winter Zhang\)](#).
- Added a check of number and type of arguments when creating `BloomFilter` index [#9623](#). [#10431 \(Winter Zhang\)](#).
- Fixed the issue when a query with `ARRAY JOIN`, `ORDER BY` and `LIMIT` may return incomplete result. This fixes [#10226](#). [#10427 \(alexey-milovidov\)](#).
- Prefer `fallback_to_stale_replicas` over `skip_unavailable_shards`. [#10422 \(Azat Khuzhin\)](#).
- Fixed wrong flattening of `Array(Tuple(...))` data types. This fixes [#10259](#). [#10390 \(alexey-milovidov\)](#).
- Fixed wrong behavior in `HashTable` that caused compilation error when trying to read `HashMap` from buffer. [#10386 \(palasonic1\)](#).
- Fixed possible Pipeline stuck error in `ConcatProcessor` which could have happened in remote query. [#10381 \(Nikolai Kochetov\)](#).
- Fixed error Pipeline stuck with `max_rows_to_group_by` and `group_by_overflow_mode = 'break'`. [#10279 \(Nikolai Kochetov\)](#).
- Fixed several bugs when some data was inserted with quorum, then deleted somehow (`DROP PARTITION, TTL`) and this leaded to the stuck of `INSERTS` or false-positive exceptions in `SELECTs`. This fixes [#9946](#). [#10188 \(Nikita Mikhaylov\)](#).
- Fixed incompatibility when versions prior to 18.12.17 are used on remote servers and newer is used on initiating server, and GROUP BY both fixed and non-fixed keys, and when two-level group by method is activated. [#3254 \(alexey-milovidov\)](#).

## Build/Testing/Packaging Improvement

- Added CA certificates to `clickhouse-server` docker image. [#10476 \(filimonov\)](#).

## 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 does not 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 does not 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 (Curtizj)
- Fix possible segfault in `MergeTreeRangeReader`, while executing `PREWHERE`. Fixes #9064. #9106 (Curtizj)
- 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 inversed 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 does not change type of column and `MODIFY` type does not 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\tb` or `a\<tab>b` and respectively, `a%5Ctb` 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 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 \(hczi\)](#)

- 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 does not 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 `(OJ)DBC` 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 does not perform redundant calculations. [#8242](#) ([achimbab](#))

- Fixed checking if a client host is allowed by matching the client host to `host_regex` 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 (filimonov)
- 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 #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 does not throw exception if `db` does not 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 does not hang indefinitely. This fixes #8126. #8128 (alexey-milovidov)
- Fix `clickhouse-copier`'s redundant contention between concurrent workers. #7816 (Ding Xiang Fei)
- Now mutations does not 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 does not 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 ([lita091](#))
- 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 does not 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 in 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 \(hczi\)](#)
- 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 \(hczi\)](#)
- 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. #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)
- 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 `Materilized 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 accomodate 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 \(hcz\)](#)
- 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 does not 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` #7766 (TLightSky)
- 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/gtest to v1.10.0. #8587 (Alexander Burmak)
- Fixed ThreadSanitizer report in `base64` library. Also updated this library to the latest version, but it does not 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 ↗ 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 \(Matwey V. Kornilov\)](#)
- 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 Release 19.17

#### ClickHouse Release 19.17.6.36, 2019-12-27

##### Bug Fix

- Fixed potential buffer overflow in decompress. Malicious user can pass fabricated compressed data that could cause read after buffer. This issue was found by Eldar Zaitov from Yandex information security team. #8404 ([alexey-milovidov](#))
- 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. #8384 ([alexey-milovidov](#))
- Fixed dictionaries with source from a clickhouse `VIEW`, now reading such dictionaries does not cause the error `There is no query.` #8351 ([Nikolai Kochetov](#))
- Fixed checking if a client host is allowed by `host_regex` specified in `users.xml`. #8241, #8342 ([Vitaly Baranov](#))
- `RENAME TABLE` for a distributed table now renames the folder containing inserted data before sending to shards. This fixes an issue with successive renames `tableA->tableB`, `tableC->tableA`. #8306 ([tavplubix](#))
- `range_hashed` external dictionaries created by DDL queries now allow ranges of arbitrary numeric types. #8275 ([alesapin](#))
- Fixed `INSERT INTO table SELECT ... FROM mysql(...)` table function. #8234 ([tavplubix](#))
- Fixed segfault in `INSERT INTO TABLE FUNCTION file()` while inserting into a file which does not exist. Now in this case file would be created and then insert would be processed. #8177 ([Olga Khvostikova](#))
- Fixed bitmapAnd error when intersecting an aggregated bitmap and a scalar bitmap. #8082 ([Yue Huang](#))
- Fixed segfault when `EXISTS` query was used without `TABLE` or `DICTIONARY` qualifier, just like `EXISTS t.` #8213 ([alexey-milovidov](#))
- 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](#))
- Fixed `DROP DICTIONARY IF EXISTS db.dict`, now it does not throw exception if `db` does not exist. #8185 ([Vitaly Baranov](#))
- If a table wasn't completely dropped because of server crash, the server will try to restore and load it #8176 ([tavplubix](#))
- Fixed a trivial count query for a distributed table if there are more than two shard local table. #8164 ([小路](#))
- Fixed bug that lead to a data race in `DB::BlockStreamProfileInfo::calculateRowsBeforeLimit()` #8143 ([Alexander Kazakov](#))

- Fixed ALTER table MOVE part executed immediately after merging the specified part, which could cause moving a part which the specified part merged into. Now it correctly moves the specified part. #8104 (Vladimir Chebotarev)
- Expressions for dictionaries can be specified as strings now. This is useful for calculation of attributes while extracting data from non-ClickHouse sources because it allows to use non-ClickHouse syntax for those expressions. #8098 (alesapin)
- Fixed a very rare race in clickhouse-copier because of an overflow in ZXid. #8088 (Ding Xiang Fei)
- Fixed the bug when after the query failed (due to “Too many simultaneous queries” for example) it would not read external tables info, and the next request would interpret this info as the beginning of the next query causing an error like Unknown packet from client. #8084 (Azat Khuzhin)
- Avoid null dereference after “Unknown packet X from server” #8071 (Azat Khuzhin)
- Restore support of all ICU locales, add the ability to apply collations for constant expressions and add language name to system.collations table. #8051 (alesapin)
- Number of streams for read from StorageFile and StorageHDFS is now limited, to avoid exceeding the memory limit. #7981 (alesapin)
- Fixed CHECK TABLE query for \*MergeTree tables without key. #7979 (alesapin)
- Removed the mutation number from a part name in case there were no mutations. This removing improved the compatibility with older versions. #8250 (alesapin)
- Fixed the bug that mutations are skipped for some attached parts due to their data\_version are larger than the table mutation version. #7812 (Zhichang Yu)
- Allow starting the server with redundant copies of parts after moving them to another device. #7810 (Vladimir Chebotarev)
- Fixed the error “Sizes of columns does not match” that might appear when using aggregate function columns. #7790 (Boris Granveaud)
- Now an exception will be thrown in case of using WITH TIES alongside LIMIT BY. And now it’s possible to use TOP with LIMIT BY. #7637 (Nikita Mikhaylov)
- Fix dictionary reload if it has invalidate\_query, which stopped updates and some exception on previous update tries. #8029 (alesapin)

## ClickHouse Release 19.17.4.11, 2019-11-22

### Backward Incompatible Change

- Using column instead of AST to store scalar subquery results for better performance. Setting enable\_scalar\_subquery\_optimization was added in 19.17 and it was enabled by default. It leads to errors like this during upgrade to 19.17.2 or 19.17.3 from previous versions. This setting was disabled by default in 19.17.4, to make possible upgrading from 19.16 and older versions without errors. #7392 (Amos Bird)

### New Feature

- Add the ability to create dictionaries with DDL queries. #7360 (alesapin)
- Make bloom\_filter type of index supporting LowCardinality and Nullable #7363 #7561 (Nikolai Kochetov)
- Add function isValidJSON to check that passed string is a valid json. #5910 #7293 (Vdimir)
- Implement arrayCompact function #7328 (Memo)

- Created function `hex` for Decimal numbers. It works like `hex(reinterpretAsString())`, but does not delete last zero bytes. [#7355 \(Mikhail Korotov\)](#)
- Add `arrayFill` and `arrayReverseFill` functions, which replace elements by other elements in front/back of them in the array. [#7380 \(hcz\)](#)
- Add `CRC32IEEE() / CRC64()` support [#7480 \(Azat Khuzhin\)](#)
- Implement `char` function similar to one in mysql [#7486 \(sundyli\)](#)
- Add `bitmapTransform` function. It transforms an array of values in a bitmap to another array of values, the result is a new bitmap [#7598 \(Zhichang Yu\)](#)
- Implemented `javaHashUTF16LE()` function [#7651 \(achimbab\)](#)
- Add `_shard_num` virtual column for the Distributed engine [#7624 \(Azat Khuzhin\)](#)

## Experimental Feature

- Support for processors (new query execution pipeline) in MergeTree. [#7181 \(Nikolai Kochetov\)](#)

## Bug Fix

- Fix incorrect float parsing in `Values` [#7817 #7870 \(tavplubix\)](#)
- Fix rare deadlock which can happen when `trace_log` is enabled. [#7838 \(filimonov\)](#)
- Prevent message duplication when producing Kafka table has any MVs selecting from it [#7265 \(Ivan\)](#)
- Support for `Array(LowCardinality(Nullable(String)))` in `IN`. Resolves [#7364 #7366 \(achimbab\)](#)
- Add handling of `SQL_TINYINT` and `SQL_BIGINT`, and fix handling of `SQL_FLOAT` data source types in ODBC Bridge. [#7491 \(Denis Glazachev\)](#)
- Fix aggregation (`avg` and `quantiles`) over empty decimal columns [#7431 \(Andrey Konyaev\)](#)
- Fix `INSERT` into Distributed with `MATERIALIZED` columns [#7377 \(Azat Khuzhin\)](#)
- Make `MOVE PARTITION` work if some parts of partition are already on destination disk or volume [#7434 \(Vladimir Chebotarev\)](#)
- Fixed bug with hardlinks failing to be created during mutations in `ReplicatedMergeTree` in multi-disk configurations. [#7558 \(Vladimir Chebotarev\)](#)
- Fixed a bug with a mutation on a MergeTree when whole part remains unchanged and best space is being found on another disk [#7602 \(Vladimir Chebotarev\)](#)
- Fixed bug with `keep_free_space_ratio` not being read from disks configuration [#7645 \(Vladimir Chebotarev\)](#)
- Fix bug with table contains only `Tuple` columns or columns with complex paths. Fixes [7541](#). [#7545 \(alesapin\)](#)
- Do not account memory for Buffer engine in `max_memory_usage` limit [#7552 \(Azat Khuzhin\)](#)
- Fix final mark usage in MergeTree tables ordered by `tuple()`. In rare cases it could lead to `Can't adjust last granule` error while select. [#7639 \(Anton Popov\)](#)
- Fix bug in mutations that have predicate with actions that require context (for example functions for json), which may lead to crashes or strange exceptions. [#7664 \(alesapin\)](#)
- Fix mismatch of database and table names escaping in `data/` and `shadow/` directories [#7575 \(Alexander Burmak\)](#)

- Support duplicated keys in RIGHT|FULL JOINS, e.g. ON t.x = u.x AND t.x = u.y. Fix crash in this case. #7586 (Artem Zuikov)
- Fix Not found column <expression> in block when joining on expression with RIGHT or FULL JOIN. #7641 (Artem Zuikov)
- One more attempt to fix infinite loop in PrettySpace format #7591 (Olga Khvostikova)
- Fix bug in concat function when all arguments were FixedString of the same size. #7635 (alesapin)
- Fixed exception in case of using 1 argument while defining S3, URL and HDFS storages. #7618 (Vladimir Chebotarev)
- Fix scope of the InterpreterSelectQuery for views with query #7601 (Azat Khuzhin)

## Improvement

- Nullable columns recognized and NULL-values handled correctly by ODBC-bridge #7402 (Vasily Nemkov)
- Write current batch for distributed send atomically #7600 (Azat Khuzhin)
- Throw an exception if we cannot detect table for column name in query. #7358 (Artem Zuikov)
- Add merge\_max\_block\_size setting to MergeTreeSettings #7412 (Artem Zuikov)
- Queries with HAVING and without GROUP BY assume group by constant. So, SELECT 1 HAVING 1 now returns a result. #7496 (Amos Bird)
- Support parsing (X,) as tuple similar to python. #7501, #7562 (Amos Bird)
- Make range function behaviors almost like pythonic one. #7518 (sundyli)
- Add constraints columns to table system.settings #7553 (Vitaly Baranov)
- Better Null format for tcp handler, so that it's possible to use select ignore(<expression>) from table format Null for perf measure via clickhouse-client #7606 (Amos Bird)
- Queries like CREATE TABLE ... AS (SELECT (1, 2)) are parsed correctly #7542 (hcz)

## Performance Improvement

- The performance of aggregation over short string keys is improved. #6243 (Alexander Kuzmenkov, Amos Bird)
- Run another pass of syntax/expression analysis to get potential optimizations after constant predicates are folded. #7497 (Amos Bird)
- Use storage meta info to evaluate trivial SELECT count() FROM table; #7510 (Amos Bird, alexey-milovidov)
- Vectorize processing arrayReduce similar to Aggregator addBatch. #7608 (Amos Bird)
- Minor improvements in performance of Kafka consumption #7475 (Ivan)

## Build/Testing/Packaging Improvement

- Add support for cross-compiling to the CPU architecture AARCH64. Refactor packager script. #7370 #7539 (Ivan)
- Unpack darwin-x86\_64 and linux-aarch64 toolchains into mounted Docker volume when building packages #7534 (Ivan)
- Update Docker Image for Binary Packager #7474 (Ivan)
- Fixed compile errors on MacOS Catalina #7585 (Ernest Poletaev)

- Some refactoring in query analysis logic: split complex class into several simple ones. [#7454](#) ([Artem Zuikov](#))
- Fix build without submodules [#7295](#) ([proller](#))
- Better `add_globs` in CMake files [#7418](#) ([Amos Bird](#))
- Remove hardcoded paths in `unwind` target [#7460](#) ([Konstantin Podshumok](#))
- Allow to use mysql format without ssl [#7524](#) ([proller](#))

## Other

- Added ANTLR4 grammar for ClickHouse SQL dialect [#7595](#) [#7596](#) ([alexey-milovidov](#))

# ClickHouse Release 19.16

ClickHouse Release 19.16.14.65, 2020-03-25

- Fixed up a bug in batched calculations of ternary logical OPs on multiple arguments (more than 10). [#8718](#) ([Alexander Kazakov](#)) This bugfix was backported to version 19.16 by a special request from Altinity.

ClickHouse Release 19.16.14.65, 2020-03-05

- Fix distributed subqueries incompatibility with older CH versions. Fixes [#7851](#) ([tabplubix](#))
- 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](#))
- Now background merges in `*MergeTree` table engines family preserve storage policy volume order more accurately.  
[#8549](#) ([Vladimir Chebotarev](#))
- 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](#))
- Allow using `MaterializedView` with subqueries above `Kafka` tables.  
[#8197](#) ([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 19.16.2.2, 2019-10-30

Backward Incompatible Change

- Add missing arity validation for count/countIf.  
[#7095](#)  
[#7298 \(Vdimir\)](#)
- Remove legacy `asterisk_left_columns_only` setting (it was disabled by default).  
[#7335 \(Artem Zuikov\)](#)
- Format strings for Template data format are now specified in files.  
[#7118 \(tavplubix\)](#)

## New Feature

- Introduce `uniqCombined64()` to calculate cardinality greater than `UINT_MAX`.  
[#7213](#),  
[#7222 \(Azat Khuzhin\)](#)
- Support Bloom filter indexes on Array columns.  
[#6984 \(achimbab\)](#)
- Add a function `getMacro(name)` that returns String with the value of corresponding `<macros>` from server configuration. [#7240 \(alexey-milovidov\)](#)
- Set two configuration options for a dictionary based on an HTTP source: `credentials` and `http-headers`. [#7092 \(Guillaume Tassery\)](#)
- Add a new ProfileEvent `Merge` that counts the number of launched background merges.  
[#7093 \(Mikhail Korotov\)](#)
- Add fullHostName function that returns a fully qualified domain name.  
[#7263](#)  
[#7291 \(sundyli\)](#)
- Add function `arraySplit` and `arrayReverseSplit` which split an array by “cut off” conditions. They are useful in time sequence handling.  
[#7294 \(hczi\)](#)
- Add new functions that return the Array of all matched indices in multiMatch family of functions.  
[#7299 \(Danila Kutenin\)](#)
- Add a new database engine `Lazy` that is optimized for storing a large number of small -Log tables. [#7171 \(Nikita Vasilev\)](#)
- Add aggregate functions `groupBitmapAnd`, `-Or`, `-Xor` for bitmap columns. [#7109 \(Zhichang Yu\)](#)
- Add aggregate function combinators `-OrNull` and `-OrDefault`, which return null or default values when there is nothing to aggregate.  
[#7331 \(hczi\)](#)

- Introduce CustomSeparated data format that supports custom escaping and delimiter rules. #7118  
([tavplubix](#))
- Support Redis as source of external dictionary. #4361 #6962 ([comunodi](#), [Anton Popov](#))

## Bug Fix

- Fix wrong query result if it has WHERE IN (SELECT ...) section and `optimize_read_in_order` is used. #7371 ([Anton Popov](#))
- Disabled MariaDB authentication plugin, which depends on files outside of project. #7140 ([Yuriy Baranov](#))
- Fix exception `Cannot convert column ... because it is constant but values of constants are different in source and result` which could rarely happen when functions `now()`, `today()`, `yesterday()`, `randConstant()` are used. #7156 ([Nikolai Kochetov](#))
- Fixed issue of using HTTP keep alive timeout instead of TCP keep alive timeout. #7351 ([Vasily Nemkov](#))
- Fixed a segmentation fault in `groupBitmapOr` (issue #7109). #7289 ([Zhichang Yu](#))
- For materialized views the commit for Kafka is called after all data were written. #7175 ([Ivan](#))
- Fixed wrong `duration_ms` value in `system.part_log` table. It was ten times off. #7172 ([Vladimir Chebotarev](#))
- A quick fix to resolve crash in LIVE VIEW table and re-enabling all LIVE VIEW tests. #7201 ([vzakaznikov](#))
- Serialize NULL values correctly in min/max indexes of MergeTree parts. #7234 ([Alexander Kuzmenkov](#))
- Don't put virtual columns to .sql metadata when table is created as `CREATE TABLE AS`. #7183 ([Ivan](#))
- Fix segmentation fault in `ATTACH PART` query. #7185 ([alesapin](#))
- Fix wrong result for some queries given by the optimization of empty IN subqueries and empty INNER/RIGHT JOIN. #7284 ([Nikolai Kochetov](#))

- Fixing AddressSanitizer error in the LIVE VIEW getHeader() method.

#7271

(vzakaznikov)

## Improvement

- Add a message in case of queue\_wait\_max\_ms wait takes place.

#7390 (Azat

Khuzhin)

- Made setting s3\_min\_upload\_part\_size table-level.

#7059 (Vladimir

Chebotarev)

- Check TTL in StorageFactory. #7304

(sundyli)

- Squash left-hand blocks in partial merge join (optimization).

#7122 (Artem

Zuikov)

- Do not allow non-deterministic functions in mutations of Replicated table engines, because this can introduce inconsistencies between replicas.

#7247 (Alexander

Kazakov)

- Disable memory tracker while converting exception stack trace to string. It can prevent the loss of error messages of type Memory limit exceeded on server, which caused the Attempt to read after eof exception on client. #7264

(Nikolai Kochetov)

- Miscellaneous format improvements. Resolves

#6033,

#2633,

#6611,

#6742

#7215

(tavplubix)

- ClickHouse ignores values on the right side of IN operator that are not convertible to the left side type. Make it work properly for compound types – Array and Tuple.

#7283 (Alexander

Kuzmenkov)

- Support missing inequalities for ASOF JOIN. It's possible to join less-or-equal variant and strict greater and less variants for ASOF column in ON syntax.

#7282 (Artem

Zuikov)

- Optimize partial merge join. #7070

(Artem Zuikov)

- Do not use more than 98K of memory in uniqCombined functions.

#7236,

#7270 (Azat

Khuzhin)

- Flush parts of right-hand joining table on disk in PartialMergeJoin (if there is not enough memory). Load data back when needed. [#7186](#)  
[\(Artem Zuikov\)](#)

## Performance Improvement

- Speed up joinGet with const arguments by avoiding data duplication.  
[#7359](#) ([Amos Bird](#))
- Return early if the subquery is empty.  
[#7007](#) ([小路](#))
- Optimize parsing of SQL expression in Values.  
[#6781](#)  
[\(tavplubix\)](#)

## Build/Testing/Packaging Improvement

- Disable some contribs for cross-compilation to Mac OS.  
[#7101](#) ([Ivan](#))
- Add missing linking with PocoXML for clickhouse\_common\_io.  
[#7200](#) ([Azat Khuzhin](#))
- Accept multiple test filter arguments in clickhouse-test.  
[#7226](#) ([Alexander Kuzmenkov](#))
- Enable musl and jemalloc for ARM. [#7300](#)  
([Amos Bird](#))
- Added --client-option parameter to `clickhouse-test` to pass additional parameters to client.  
[#7277](#) ([Nikolai Kochetov](#))
- Preserve existing configs on rpm package upgrade.  
[#7103](#)  
([filimonov](#))
- Fix errors detected by PVS. [#7153](#) ([Artem Zuikov](#))
- Fix build for Darwin. [#7149](#)  
([Ivan](#))
- glibc 2.29 compatibility. [#7142](#) ([Amos Bird](#))
- Make sure dh\_clean does not touch potential source files.  
[#7205](#) ([Amos Bird](#))
- Attempt to avoid conflict when updating from altinity rpm - it has config file packaged separately in clickhouse-server-common. [#7073](#)  
([filimonov](#))

- Optimize some header files for faster rebuilds.  
[#7212](#),  
[#7231 \(Alexander Kuzmenkov\)](#)
- Add performance tests for Date and DateTime. [#7332 \(Vasily Nemkov\)](#)
- Fix some tests that contained non-deterministic mutations.  
[#7132 \(Alexander Kazakov\)](#)
- Add build with MemorySanitizer to CI. [#7066 \(Alexander Kuzmenkov\)](#)
- Avoid use of uninitialized values in MetricsTransmitter.  
[#7158 \(Azat Khuzhin\)](#)
- Fix some issues in Fields found by MemorySanitizer.  
[#7135](#),  
[#7179 \(Alexander Kuzmenkov\), #7376 \(Amos Bird\)](#)
- Fix undefined behavior in murmurhash32. [#7388 \(Amos Bird\)](#)
- Fix undefined behavior in StoragesInfoStream. [#7384 \(tavplubix\)](#)
- Fixed constant expressions folding for external database engines (MySQL, ODBC, JDBC). In previous versions it wasn't working for multiple constant expressions and was not working at all for Date, DateTime and UUID. This fixes [#7245 #7252 \(alexey-milovidov\)](#)
- Fixing ThreadSanitizer data race error in the LIVE VIEW when accessing no\_users\_thread variable.  
[#7353 \(vzakaznikov\)](#)
- Get rid of malloc symbols in libcommon  
[#7134](#),  
[#7065 \(Amos Bird\)](#)
- Add global flag ENABLE\_LIBRARIES for disabling all libraries.  
[#7063 \(proller\)](#)

## Code Cleanup

- Generalize configuration repository to prepare for DDL for Dictionaries. [#7155 \(alesapin\)](#)
- Parser for dictionaries DDL without any semantic.  
[#7209 \(alesapin\)](#)

- Split ParserCreateQuery into different smaller parsers.  
[#7253](#)  
(alesapin)
- Small refactoring and renaming near external dictionaries.  
[#7111](#)  
(alesapin)
- Refactor some code to prepare for role-based access control. [#7235](#) ([Vitaly Baranov](#))
- Some improvements in DatabaseOrdinary code.  
[#7086](#) ([Nikita Vasilev](#))
- Do not use iterators in find() and emplace() methods of hash tables.  
[#7026](#) ([Alexander Kuzmenkov](#))
- Fix getMultipleValuesFromConfig in case when parameter root is not empty. [#7374](#) ([Mikhail Korotov](#))
- Remove some copy-paste (TemporaryFile and TemporaryFileStream)  
[#7166](#) ([Artem Zuikov](#))
- Improved code readability a little bit (MergeTreeData::getActiveContainingPart).  
[#7361](#) ([Vladimir Chebotarev](#))
- Wait for all scheduled jobs, which are using local objects, if `ThreadPool::schedule(...)` throws an exception. Rename `ThreadPool::schedule(...)` to `ThreadPool::scheduleOrThrowOnError(...)` and fix comments to make obvious that it may throw.  
[#7350](#)  
(tavplubix)

## ClickHouse Release 19.15

### ClickHouse Release 19.15.4.10, 2019-10-31

#### Bug Fix

- Added handling of SQL\_TINYINT and SQL\_BIGINT, and fix handling of SQL\_FLOAT data source types in ODBC Bridge.  
[#7491](#) ([Denis Glazachev](#))
- Allowed to have some parts on destination disk or volume in MOVE PARTITION.  
[#7434](#) ([Vladimir Chebotarev](#))
- Fixed NULL-values in nullable columns through ODBC-bridge.  
[#7402](#) ([Vasily Nemkov](#))
- Fixed INSERT into Distributed non local node with MATERIALIZED columns.  
[#7377](#) ([Azat Khuzhin](#))
- Fixed function getMultipleValuesFromConfig.  
[#7374](#) ([Mikhail Korotov](#))

- Fixed issue of using HTTP keep alive timeout instead of TCP keep alive timeout.  
[#7351 \(Vasily Nemkov\)](#)
- Wait for all jobs to finish on exception (fixes rare segfaults).  
[#7350 \(tavplubix\)](#)
- Don't push to MVs when inserting into Kafka table.  
[#7265 \(Ivan\)](#)
- Disable memory tracker for exception stack.  
[#7264 \(Nikolai Kochetov\)](#)
- Fixed bad code in transforming query for external database.  
[#7252 \(alexey-milovidov\)](#)
- Avoid use of uninitialized values in MetricsTransmitter.  
[#7158 \(Azat Khuzhin\)](#)
- Added example config with macros for tests ([alexey-milovidov](#))

## ClickHouse Release 19.15.3.6, 2019-10-09

### Bug Fix

- Fixed bad\_variant in hashed dictionary.  
[\(alesapin\)](#)
- Fixed up bug with segmentation fault in ATTACH PART query.  
[\(alesapin\)](#)
- Fixed time calculation in MergeTreeData.  
[\(Vladimir Chebotarev\)](#)
- Commit to Kafka explicitly after the writing is finalized.  
[#7175 \(Ivan\)](#)
- Serialize NULL values correctly in min/max indexes of MergeTree parts.  
[#7234 \(Alexander Kuzmenkov\)](#)

## ClickHouse Release 19.15.2.2, 2019-10-01

### New Feature

- Tiered storage: support to use multiple storage volumes for tables with MergeTree engine. It's possible to store fresh data on SSD and automatically move old data to HDD. ([example](#)). [#4918 \(Igr\)](#) [#6489 \(alesapin\)](#)
- Add table function `input` for reading incoming data in `INSERT SELECT` query. [#5450 \(palasonic1\)](#) [#6832 \(Anton Popov\)](#)
- Add a `sparse_hashed` dictionary layout, that is functionally equivalent to the `hashed` layout, but is more memory efficient. It uses about twice as less memory at the cost of slower value retrieval. [#6894 \(Azat Khuzhin\)](#)
- Implement ability to define list of users for access to dictionaries. Only current connected database using. [#6907 \(Guillaume Tassery\)](#)
- Add `LIMIT` option to `SHOW` query. [#6944 \(Philipp Malkovsky\)](#)
- Add `bitmapSubsetLimit(bitmap, range_start, limit)` function, that returns subset of the smallest `limit` values in set that is no smaller than `range_start`. [#6957 \(Zhichang Yu\)](#)

- Add `bitmapMin` and `bitmapMax` functions. #6970 (Zhichang Yu)

- Add function `repeat` related to issue-6648 #6999 (flynn)

## Experimental Feature

- Implement (in memory) Merge Join variant that does not change current pipeline. Result is partially sorted by merge key. Set `partial_merge_join = 1` to use this feature. The Merge Join is still in development. #6940 (Artem Zuikov)
- Add S3 engine and table function. It is still in development (no authentication support yet). #5596 (Vladimir Chebotarev)

## Improvement

- Every message read from Kafka is inserted atomically. This resolves almost all known issues with Kafka engine. #6950 (Ivan)
- Improvements for failover of Distributed queries. Shorten recovery time, also it is now configurable and can be seen in `system.clusters`. #6399 (Vasily Nemkov)
- Support numeric values for Enums directly in `IN` section. #6766 #6941 (dimarub2000)
- Support (optional, disabled by default) redirects on URL storage. #6914 (maqroll)
- Add information message when client with an older version connects to a server. #6893 (Philipp Malkovsky)
- Remove maximum backoff sleep time limit for sending data in Distributed tables #6895 (Azat Khuzhin)
- Add ability to send profile events (counters) with cumulative values to graphite. It can be enabled under `<events_cumulative>` in server `config.xml`. #6969 (Azat Khuzhin)
- Add automatically cast type `T` to `LowCardinality(T)` while inserting data in column of type `LowCardinality(T)` in Native format via HTTP. #6891 (Nikolai Kochetov)
- Add ability to use function `hex` without using `reinterpretAsString` for `Float32`, `Float64`. #7024 (Mikhail Korotov)

## Build/Testing/Packaging Improvement

- Add gdb-index to clickhouse binary with debug info. It will speed up startup time of gdb. #6947 (alesapin)
- Speed up deb packaging with patched dpkg-deb which uses pigz. #6960 (alesapin)
- Set `enable_fuzzing = 1` to enable libfuzzer instrumentation of all the project code. #7042 (kyprizel)
- Add split build smoke test in CI. #7061 (alesapin)
- Add build with MemorySanitizer to CI. #7066 (Alexander Kuzmenkov)
- Replace `libsparsehash` with `sparsehash-c11` #6965 (Azat Khuzhin)

## Bug Fix

- Fixed performance degradation of index analysis on complex keys on large tables. This fixes #6924. #7075 (alexey-milovidov)
- Fix logical error causing segfaults when selecting from Kafka empty topic. #6909 (Ivan)
- Fix too early MySQL connection close in `MySQLBlockInputStream.cpp`. #6882 (Clément Rodriguez)
- Returned support for very old Linux kernels (fix #6841) #6853 (alexey-milovidov)

- Fix possible data loss in `insert select` query in case of empty block in input stream. #6834 #6862 #6911 ([Nikolai Kochetov](#))
- Fix for function `ArrayEnumerateUniqRanked` with empty arrays in params #6928 ([proller](#))
- Fix complex queries with array joins and global subqueries. #6934 ([Ivan](#))
- Fix `Unknown identifier` error in `ORDER BY` and `GROUP BY` with multiple `JOINS` #7022 ([Artem Zuikov](#))
- Fixed `MSan` warning while executing function with `LowCardinality` argument. #7062 ([Nikolai Kochetov](#))

## Backward Incompatible Change

- Changed serialization format of `bitmap*` aggregate function states to improve performance. Serialized states of `bitmap*` from previous versions cannot be read. #6908 ([Zhichang Yu](#))

# ClickHouse Release 19.14

## ClickHouse Release 19.14.7.15, 2019-10-02

### Bug Fix

- This release also contains all bug fixes from 19.11.12.69.
- Fixed compatibility for distributed queries between 19.14 and earlier versions. This fixes #7068. #7069 ([alexey-milovidov](#))

## ClickHouse Release 19.14.6.12, 2019-09-19

### Bug Fix

- Fix for function `ArrayEnumerateUniqRanked` with empty arrays in params. #6928 ([proller](#))
- Fixed subquery name in queries with `ARRAY JOIN` and `GLOBAL IN subquery` with alias. Use subquery alias for external table name if it is specified. #6934 ([Ivan](#))

### Build/Testing/Packaging Improvement

- Fix `flapping` test `00715_fetch_merged_or_mutated_part_zookeeper` by rewriting it to a shell scripts because it needs to wait for mutations to apply. #6977 ([Alexander Kazakov](#))
- Fixed UBSan and MemSan failure in function `groupUniqArray` with empty array argument. It was caused by placing of empty `PaddedPODArray` into hash table zero cell because constructor for zero cell value was not called. #6937 ([Amos Bird](#))

## ClickHouse Release 19.14.3.3, 2019-09-10

### New Feature

- `WITH FILL` modifier for `ORDER BY`. (continuation of #5069) #6610 ([Anton Popov](#))
- `WITH TIES` modifier for `LIMIT`. (continuation of #5069) #6610 ([Anton Popov](#))
- Parse unquoted `NULL` literal as `NULL` (if setting `format_csv_unquoted_null_literal_as_null=1`). Initialize null fields with default values if data type of this field is not nullable (if setting `input_format_null_as_default=1`). #5990 #6055 ([tavplubix](#))
- Support for wildcards in paths of table functions `file` and `hdfs`. If the path contains wildcards, the table will be readonly. Example of usage: `select * from hdfs('hdfs://hdfs1:9000/some_dir/another_dir/*/*{0..9}{0..9}')` and `select * from file('some_dir/{some_file,another_file,yet_another}.tsv', 'TSV', 'value UInt32')`. #6092 ([Olga Khvostikova](#))

- New `system.metric_log` table which stores values of `system.events` and `system.metrics` with specified time interval. #6363 #6467 (Nikita Mikhaylov) #6530 (alexey-milovidov)
- Allow to write ClickHouse text logs to `system.text_log` table. #6037 #6103 (Nikita Mikhaylov) #6164 (alexey-milovidov)
- Show private symbols in stack traces (this is done via parsing symbol tables of ELF files). Added information about file and line number in stack traces if debug info is present. Speedup symbol name lookup with indexing symbols present in program. Added new SQL functions for introspection: `demangle` and `addressToLine`. Renamed function `symbolizeAddress` to `addressToSymbol` for consistency. Function `addressToSymbol` will return mangled name for performance reasons and you have to apply `demangle`. Added setting `allow_introspection_functions` which is turned off by default. #6201 (alexey-milovidov)
- Table function `values` (the name is case-insensitive). It allows to read from `VALUES` list proposed in #5984. Example: `SELECT * FROM VALUES('a UInt64, s String', (1, 'one'), (2, 'two'), (3, 'three'))` #6217. #6209 (dimarub2000)
- Added an ability to alter storage settings. Syntax: `ALTER TABLE <table> MODIFY SETTING <setting> = <value>`. #6366 #6669 #6685 (alesapin)
- Support for removing of detached parts. Syntax: `ALTER TABLE <table_name> DROP DETACHED PART '<part_id>'`. #6158 (tavplubix)
- Table constraints. Allows to add constraint to table definition which will be checked at insert. #5273 (Gleb Novikov) #6652 (alexey-milovidov)
- Support for cascaded materialized views. #6324 (Amos Bird)
- Turn on query profiler by default to sample every query execution thread once a second. #6283 (alexey-milovidov)
- Input format `ORC`. #6454 #6703 (akonyaev90)
- Added two new functions: `sigmoid` and `tanh` (that are useful for machine learning applications). #6254 (alexey-milovidov)
- Function `hasToken(haystack, token)`, `hasTokenCaseInsensitive(haystack, token)` to check if given token is in haystack. Token is a maximal length substring between two non alphanumeric ASCII characters (or boundaries of haystack). Token must be a constant string. Supported by `tokenbf_v1` index specialization. #6596, #6662 (Vasily Nemkov)
- New function `neighbor(value, offset[, default_value])`. Allows to reach prev/next value within column in a block of data. #5925 (Alex Krash) 6685365ab8c5b74f9650492c88a012596eb1b0c6 341e2e4587a18065c2da1ca888c73389f48ce36c Alexey Milovidov
- Created a function `currentUser()`, returning login of authorized user. Added alias `user()` for compatibility with MySQL. #6470 (Alex Krash)
- New aggregate functions `quantilesExactInclusive` and `quantilesExactExclusive` which were proposed in #5885. #6477 (dimarub2000)
- Function `bitmapRange(bitmap, range_begin, range_end)` which returns new set with specified range (not include the `range_end`). #6314 (Zhichang Yu)
- Function `geohashesInBox(longitude_min, latitude_min, longitude_max, latitude_max, precision)` which creates array of precision-long strings of geohash-boxes covering provided area. #6127 (Vasily Nemkov)
- Implement support for `INSERT` query with `Kafka` tables. #6012 (Ivan)
- Added support for `_partition` and `_timestamp` virtual columns to Kafka engine. #6400 (Ivan)

- Possibility to remove sensitive data from `query_log`, server logs, process list with regexp-based rules. #5710 (filimonov)

## Experimental Feature

- Input and output data format `Template`. It allows to specify custom format string for input and output. #4354 #6727 (tavplubix)
- Implementation of `LIVE VIEW` tables that were originally proposed in #2898, prepared in #3925, and then updated in #5541. See #5541 for detailed description. #5541 (vzakaznikov) #6425 (Nikolai Kochetov) #6656 (vzakaznikov) Note that `LIVE VIEW` feature may be removed in next versions.

## Bug Fix

- This release also contains all bug fixes from 19.13 and 19.11.
- Fix segmentation fault when the table has skip indices and vertical merge happens. #6723 (alesapin)
- Fix per-column TTL with non-trivial column defaults. Previously in case of force TTL merge with `OPTIMIZE ... FINAL` query, expired values was replaced by type defaults instead of user-specified column defaults. #6796 (Anton Popov)
- Fix Kafka messages duplication problem on normal server restart. #6597 (Ivan)
- Fixed infinite loop when reading Kafka messages. Do not pause/resume consumer on subscription at all - otherwise it may get paused indefinitely in some scenarios. #6354 (Ivan)
- Fix `Key expression contains comparison between inconvertible types` exception in `bitmapContains` function. #6136 #6146 #6156 (dimarub2000)
- Fix segfault with enabled `optimize_skip_unused_shards` and missing sharding key. #6384 (Anton Popov)
- Fixed wrong code in mutations that may lead to memory corruption. Fixed segfault with read of address `0x14c0` that may happed due to concurrent `DROP TABLE` and `SELECT` from `system.parts` or `system.parts_columns`. Fixed race condition in preparation of mutation queries. Fixed deadlock caused by `OPTIMIZE` of Replicated tables and concurrent modification operations like `ALTERs`. #6514 (alexey-milovidov)
- Removed extra verbose logging in MySQL interface #6389 (alexey-milovidov)
- Return the ability to parse boolean settings from 'true' and 'false' in the configuration file. #6278 (alesapin)
- Fix crash in `quantile` and `median` function over `Nullable(Decimal128)`. #6378 (Artem Zuikov)
- Fixed possible incomplete result returned by `SELECT` query with `WHERE` condition on primary key contained conversion to `Float` type. It was caused by incorrect checking of monotonicity in `toFloat` function. #6248 #6374 (dimarub2000)
- Check `max_expanded_ast_elements` setting for mutations. Clear mutations after `TRUNCATE TABLE`. #6205 (Winter Zhang)
- Fix JOIN results for key columns when used with `join_use_nulls`. Attach Nulls instead of columns defaults. #6249 (Artem Zuikov)
- Fix for skip indices with vertical merge and alter. Fix for `Bad size of marks file` exception. #6594 #6713 (alesapin)
- Fix rare crash in `ALTER MODIFY COLUMN` and vertical merge when one of merged/altered parts is empty (0 rows) #6746 #6780 (alesapin)

- Fixed bug in conversion of `LowCardinality` types in `AggregateFunctionFactory`. This fixes #6257. #6281 ([Nikolai Kochetov](#))
- Fix wrong behavior and possible segfaults in `topK` and `topKWeighted` aggregated functions. #6404 ([Anton Popov](#))
- Fixed unsafe code around `getIdentifier` function. #6401 #6409 ([alexey-milovidov](#))
- Fixed bug in MySQL wire protocol (is used while connecting to ClickHouse from MySQL client). Caused by heap buffer overflow in `PacketPayloadWriteBuffer`. #6212 ([Yuriy Baranov](#))
- Fixed memory leak in `bitmapSubsetInRange` function. #6819 ([Zhichang Yu](#))
- Fix rare bug when mutation executed after granularity change. #6816 ([alesapin](#))
- Allow protobuf message with all fields by default. #6132 ([Vitaly Baranov](#))
- Resolve a bug with `nullIf` function when we send a `NULL` argument on the second argument. #6446 ([Guillaume Tassery](#))
- Fix rare bug with wrong memory allocation/deallocation in complex key cache dictionaries with string fields which leads to infinite memory consumption (looks like memory leak). Bug reproduces when string size was a power of two starting from eight (8, 16, 32, etc). #6447 ([alesapin](#))
- Fixed Gorilla encoding on small sequences which caused exception `Cannot write after end of buffer`. #6398 #6444 ([Vasily Nemkov](#))
- Allow to use not nullable types in JOINS with `join_use_nulls` enabled. #6705 ([Artem Zuikov](#))
- Disable `Poco::AbstractConfiguration` substitutions in query in `clickhouse-client`. #6706 ([alexey-milovidov](#))
- Avoid deadlock in `REPLACE PARTITION`. #6677 ([alexey-milovidov](#))
- Using `arrayReduce` for constant arguments may lead to segfault. #6242 #6326 ([alexey-milovidov](#))
- Fix inconsistent parts which can appear if replica was restored after `DROP PARTITION`. #6522 #6523 ([tavplubix](#))
- Fixed hang in `JSONExtractRaw` function. #6195 #6198 ([alexey-milovidov](#))
- Fix bug with incorrect skip indices serialization and aggregation with adaptive granularity. #6594. #6748 ([alesapin](#))
- Fix WITH ROLLUP and WITH CUBE modifiers of GROUP BY with two-level aggregation. #6225 ([Anton Popov](#))
- Fix bug with writing secondary indices marks with adaptive granularity. #6126 ([alesapin](#))
- Fix initialization order while server startup. Since `StorageMergeTree::background_task_handle` is initialized in `startup()` the `MergeTreeBlockOutputStream::write()` may try to use it before initialization. Just check if it is initialized. #6080 ([Ivan](#))
- Clearing the data buffer from the previous read operation that was completed with an error. #6026 ([Nikolay](#))
- Fix bug with enabling adaptive granularity when creating a new replica for Replicated\*MergeTree table. #6394 #6452 ([alesapin](#))
- Fixed possible crash during server startup in case of exception happened in `libunwind` during exception at access to uninitialized `ThreadStatus` structure. #6456 ([Nikita Mikhaylov](#))
- Fix crash in `yandexConsistentHash` function. Found by fuzz test. #6304 #6305 ([alexey-milovidov](#))

- Fixed the possibility of hanging queries when server is overloaded and global thread pool becomes near full. This have higher chance to happen on clusters with large number of shards (hundreds), because distributed queries allocate a thread per connection to each shard. For example, this issue may reproduce if a cluster of 330 shards is processing 30 concurrent distributed queries. This issue affects all versions starting from 19.2. [#6301 \(alexey-milovidov\)](#)
- Fixed logic of `arrayEnumerateUniqRanked` function. [#6423 \(alexey-milovidov\)](#)
- Fix segfault when decoding symbol table. [#6603 \(Amos Bird\)](#)
- Fixed irrelevant exception in cast of `LowCardinalityNullable` to not-`Nullable` column in case if it does not contain Nulls (e.g. in query like `SELECT CAST(CAST('Hello' AS LowCardinalityNullable(String))) AS String`). [#6094 #6119 \(Nikolai Kochetov\)](#)
- Removed extra quoting of description in `system.settings` table. [#6696 #6699 \(alexey-milovidov\)](#)
- Avoid possible deadlock in `TRUNCATE` of Replicated table. [#6695 \(alexey-milovidov\)](#)
- Fix reading in order of sorting key. [#6189 \(Anton Popov\)](#)
- Fix `ALTER TABLE ... UPDATE` query for tables with `enable_mixed_granularity_parts=1`. [#6543 \(alesapin\)](#)
- Fix bug opened by [#4405](#) (since 19.4.0). Reproduces in queries to Distributed tables over MergeTree tables when we does not query any columns (`SELECT 1`). [#6236 \(alesapin\)](#)
- Fixed overflow in integer division of signed type to unsigned type. The behaviour was exactly as in C or C++ language (integer promotion rules) that may be surprising. Please note that the overflow is still possible when dividing large signed number to large unsigned number or vice-versa (but that case is less usual). The issue existed in all server versions. [#6214 #6233 \(alexey-milovidov\)](#)
- Limit maximum sleep time for throttling when `max_execution_speed` or `max_execution_speed_bytes` is set. Fixed false errors like `Estimated query execution time (inf seconds) is too long.` [#5547 #6232 \(alexey-milovidov\)](#)
- Fixed issues about using `MATERIALIZED` columns and aliases in `MaterializedView`. [#448 #3484 #3450 #2878 #2285 #3796 \(Amos Bird\)](#) [#6316 \(alexey-milovidov\)](#)
- Fix `FormatFactory` behaviour for input streams which are not implemented as processor. [#6495 \(Nikolai Kochetov\)](#)
- Fixed typo. [#6631 \(Alex Ryndin\)](#)
- Typo in the error message ( is -> are ). [#6839 \(Denis Zhuravlev\)](#)
- Fixed error while parsing of columns list from string if type contained a comma (this issue was relevant for `File`, `URL`, `HDFS` storages) [#6217. #6209 \(dimarub2000\)](#)

## Security Fix

- This release also contains all bug security fixes from 19.13 and 19.11.
- Fixed the possibility of a fabricated query to cause server crash due to stack overflow in SQL parser. Fixed the possibility of stack overflow in Merge and Distributed tables, materialized views and conditions for row-level security that involve subqueries. [#6433 \(alexey-milovidov\)](#)

## Improvement

- Correct implementation of ternary logic for AND/OR. [#6048 \(Alexander Kazakov\)](#)

- Now values and rows with expired TTL will be removed after `OPTIMIZE ... FINAL` query from old parts without TTL infos or with outdated TTL infos, e.g. after `ALTER ... MODIFY TTL` query. Added queries `SYSTEM STOP/START TTL MERGES` to disallow/allow assign merges with TTL and filter expired values in all merges. [#6274 \(Anton Popov\)](#)
- Possibility to change the location of ClickHouse history file for client using `CLOCKHOUSE_HISTORY_FILE` env. [#6840 \(filimonov\)](#)
- Remove `dry_run` flag from `InterpreterSelectQuery`. ... [#6375 \(Nikolai Kochetov\)](#)
- Support `ASOF JOIN` with `ON` section. [#6211 \(Artem Zuiakov\)](#)
- Better support of skip indexes for mutations and replication. Support for `MATERIALIZE/CLEAR INDEX ... IN PARTITION` query. `UPDATE x = x` recalculates all indices that use column `x`. [#5053 \(Nikita Vasilev\)](#)
- Allow to `ATTACH` live views (for example, at the server startup) regardless to `allow_experimental_live_view` setting. [#6754 \(alexey-milovidov\)](#)
- For stack traces gathered by query profiler, do not include stack frames generated by the query profiler itself. [#6250 \(alexey-milovidov\)](#)
- Now table functions `values`, `file`, `url`, `hdfs` have support for ALIAS columns. [#6255 \(alexey-milovidov\)](#)
- Throw an exception if `config.d` file does not have the corresponding root element as the config file. [#6123 \(dimarub2000\)](#)
- Print extra info in exception message for no space left on device [#6182](#), [#6252](#) [#6352 \(tavplubix\)](#)
- When determining shards of a `Distributed` table to be covered by a read query (for `optimize_skip_unused_shards = 1`) ClickHouse now checks conditions from both `prewhere` and `where` clauses of select statement. [#6521 \(Alexander Kazakov\)](#)
- Enabled `SIMDJSON` for machines without AVX2 but with SSE 4.2 and PCLMUL instruction set. [#6285](#) [#6320 \(alexey-milovidov\)](#)
- ClickHouse can work on filesystems without `O_DIRECT` support (such as ZFS and Btrfs) without additional tuning. [#4449](#) [#6730 \(alexey-milovidov\)](#)
- Support push down predicate for final subquery. [#6120 \(TCeason\)](#) [#6162 \(alexey-milovidov\)](#)
- Better `JOIN ON` keys extraction [#6131 \(Artem Zuiakov\)](#)
- Updated `SIMDJSON`. [#6285](#). [#6306 \(alexey-milovidov\)](#)
- Optimize selecting of smallest column for `SELECT count()` query. [#6344 \(Amos Bird\)](#)
- Added `strict` parameter in `windowFunnel()`. When the `strict` is set, the `windowFunnel()` applies conditions only for the unique values. [#6548 \(achimbab\)](#)
- Safer interface of `mysqlxx::Pool`. [#6150 \(avasiliev\)](#)
- Options line size when executing with `--help` option now corresponds with terminal size. [#6590 \(dimarub2000\)](#)
- Disable “read in order” optimization for aggregation without keys. [#6599 \(Anton Popov\)](#)
- HTTP status code for `INCORRECT_DATA` and `TYPE_MISMATCH` error codes was changed from default 500 Internal Server Error to 400 Bad Request. [#6271 \(Alexander Rodin\)](#)
- Move `Join` object from `ExpressionAction` into `AnalyzedJoin`. `ExpressionAnalyzer` and `ExpressionAction` do not know about `Join` class anymore. Its logic is hidden by `AnalyzedJoin` iface. [#6801 \(Artem Zuiakov\)](#)

- Fixed possible deadlock of distributed queries when one of shards is localhost but the query is sent via network connection. [#6759](#) ([alexey-milovidov](#))
- Changed semantic of multiple tables `RENAME` to avoid possible deadlocks. [#6757](#). [#6756](#) ([alexey-milovidov](#))
- Rewritten MySQL compatibility server to prevent loading full packet payload in memory. Decreased memory consumption for each connection to approximately `2 * DBMS_DEFAULT_BUFFER_SIZE` (read/write buffers). [#5811](#) ([Yuriy Baranov](#))
- Move AST alias interpreting logic out of parser that does not have to know anything about query semantics. [#6108](#) ([Artem Zuikov](#))
- Slightly more safe parsing of `NamesAndTypesList`. [#6408](#). [#6410](#) ([alexey-milovidov](#))
- `clickhouse-copier`: Allow use `where_condition` from config with `partition_key` alias in query for checking partition existence (Earlier it was used only in reading data queries). [#6577](#) ([proller](#))
- Added optional message argument in `throwIf`. ([#5772](#)) [#6329](#) ([Vdimir](#))
- Server exception got while sending insertion data is now being processed in client as well. [#5891](#) [#6711](#) ([dimarub2000](#))
- Added a metric `DistributedFilesToInsert` that shows the total number of files in filesystem that are selected to send to remote servers by Distributed tables. The number is summed across all shards. [#6600](#) ([alexey-milovidov](#))
- Move most of JOINS prepare logic from `ExpressionAction/ExpressionAnalyzer` to `AnalyzedJoin`. [#6785](#) ([Artem Zuikov](#))
- Fix TSan warning ‘lock-order-inversion’. [#6740](#) ([Vasily Nemkov](#))
- Better information messages about lack of Linux capabilities. Logging fatal errors with “fatal” level, that will make it easier to find in `system.text_log`. [#6441](#) ([alexey-milovidov](#))
- When enable dumping temporary data to the disk to restrict memory usage during `GROUP BY`, `ORDER BY`, it didn’t check the free disk space. The fix add a new setting `min_free_disk_space`, when the free disk space is smaller then the threshold, the query will stop and throw `ErrorCodes::NOT_ENOUGH_SPACE`. [#6678](#) ([Weiqing Xu](#)) [#6691](#) ([alexey-milovidov](#))
- Removed recursive rwlock by thread. It makes no sense, because threads are reused between queries. `SELECT` query may acquire a lock in one thread, hold a lock from another thread and exit from first thread. In the same time, first thread can be reused by `DROP` query. This will lead to false “Attempt to acquire exclusive lock recursively” messages. [#6771](#) ([alexey-milovidov](#))
- Split `ExpressionAnalyzer.appendJoin()`. Prepare a place in `ExpressionAnalyzer` for `MergeJoin`. [#6524](#) ([Artem Zuikov](#))
- Added `mysql_native_password` authentication plugin to MySQL compatibility server. [#6194](#) ([Yuriy Baranov](#))
- Less number of `clock_gettime` calls; fixed ABI compatibility between debug/release in `Allocator` (insignificant issue). [#6197](#) ([alexey-milovidov](#))
- Move `collectUsedColumns` from `ExpressionAnalyzer` to `SyntaxAnalyzer`. `SyntaxAnalyzer` makes `required_source_columns` itself now. [#6416](#) ([Artem Zuikov](#))
- Add setting `joined_subquery_requires_alias` to require aliases for subselects and table functions in `FROM` that more than one table is present (i.e. queries with JOINS). [#6733](#) ([Artem Zuikov](#))
- Extract `GetAggregatesVisitor` class from `ExpressionAnalyzer`. [#6458](#) ([Artem Zuikov](#))

- `system.query_log`: change data type of `type` column to `Enum`. [#6265 \(Nikita Mikhaylov\)](#)
- Static linking of `sha256_password` authentication plugin. [#6512 \(Yuriy Baranov\)](#)
- Avoid extra dependency for the setting `compile` to work. In previous versions, the user may get error like `cannot open crt1.o, unable to find library -lc` etc. [#6309 \(alexey-milovidov\)](#)
- More validation of the input that may come from malicious replica. [#6303 \(alexey-milovidov\)](#)
- Now `clickhouse-obfuscator` file is available in `clickhouse-client` package. In previous versions it was available as `clickhouse obfuscator` (with whitespace). [#5816 #6609 \(dimarub2000\)](#)
- Fixed deadlock when we have at least two queries that read at least two tables in different order and another query that performs DDL operation on one of tables. Fixed another very rare deadlock. [#6764 \(alexey-milovidov\)](#)
- Added `os_thread_ids` column to `system.processes` and `system.query_log` for better debugging possibilities. [#6763 \(alexey-milovidov\)](#)
- A workaround for PHP `mysqlnd` extension bugs which occur when `sha256_password` is used as a default authentication plugin (described in [#6031](#)). [#6113 \(Yuriy Baranov\)](#)
- Remove unneeded place with changed nullability columns. [#6693 \(Artem Zuikov\)](#)
- Set default value of `queue_max_wait_ms` to zero, because current value (five seconds) makes no sense. There are rare circumstances when this settings has any use. Added settings `replace_running_query_max_wait_ms`, `kafka_max_wait_ms` and `connection_pool_max_wait_ms` for disambiguation. [#6692 \(alexey-milovidov\)](#)
- Extract `SelectQueryExpressionAnalyzer` from `ExpressionAnalyzer`. Keep the last one for non-select queries. [#6499 \(Artem Zuikov\)](#)
- Removed duplicating input and output formats. [#6239 \(Nikolai Kochetov\)](#)
- Allow user to override `poll_interval` and `idle_connection_timeout` settings on connection. [#6230 \(alexey-milovidov\)](#)
- `MergeTree` now has an additional option `ttl_only_drop_parts` (disabled by default) to avoid partial pruning of parts, so that they dropped completely when all the rows in a part are expired. [#6191 \(Sergi Vladyskin\)](#)
- Type checks for set index functions. Throw exception if function got a wrong type. This fixes fuzz test with UBSan. [#6511 \(Nikita Vasilev\)](#)

## Performance Improvement

- Optimize queries with `ORDER BY` expressions clause, where expressions have coinciding prefix with sorting key in `MergeTree` tables. This optimization is controlled by `optimize_read_in_order` setting. [#6054 #6629 \(Anton Popov\)](#)
- Allow to use multiple threads during parts loading and removal. [#6372 #6074 #6438 \(alexey-milovidov\)](#)
- Implemented batch variant of updating aggregate function states. It may lead to performance benefits. [#6435 \(alexey-milovidov\)](#)
- Using `FastOps` library for functions `exp`, `log`, `sigmoid`, `tanh`. `FastOps` is a fast vector math library from Michael Parakhin (Yandex CTO). Improved performance of `exp` and `log` functions more than 6 times. The functions `exp` and `log` from `Float32` argument will return `Float32` (in previous versions they always return `Float64`). Now `exp(nan)` may return `inf`. The result of `exp` and `log` functions may be not the nearest machine representable number to the true answer. [#6254 \(alexey-milovidov\)](#) Using Danila Kutenin variant to make `fastops` working [#6317 \(alexey-milovidov\)](#)

- Disable consecutive key optimization for UInt8/16. #6298 #6701 (akuzm)
- Improved performance of simdjson library by getting rid of dynamic allocation in ParsedJson::Iterator. #6479 (Vitaly Baranov)
- Pre-fault pages when allocating memory with mmap(). #6667 (akuzm)
- Fix performance bug in Decimal comparison. #6380 (Artem Zuikov)

## Build/Testing/Packaging Improvement

- Remove Compiler (runtime template instantiation) because we've win over it's performance. #6646 (alexey-milovidov)
- Added performance test to show degradation of performance in gcc-9 in more isolated way. #6302 (alexey-milovidov)
- Added table function numbers\_mt, which is multithreaded version of numbers. Updated performance tests with hash functions. #6554 (Nikolai Kochetov)
- Comparison mode in clickhouse-benchmark #6220 #6343 (dimarub2000)
- Best effort for printing stack traces. Also added SIGPROF as a debugging signal to print stack trace of a running thread. #6529 (alexey-milovidov)
- Every function in its own file, part 10. #6321 (alexey-milovidov)
- Remove doubled const TABLE\_IS\_READ\_ONLY. #6566 (filimonov)
- Formatting changes for StringHashMap PR #5417. #6700 (akuzm)
- Better subquery for join creation in ExpressionAnalyzer. #6824 (Artem Zuikov)
- Remove a redundant condition (found by PVS Studio). #6775 (akuzm)
- Separate the hash table interface for ReverseIndex. #6672 (akuzm)
- Refactoring of settings. #6689 (alesapin)
- Add comments for set index functions. #6319 (Nikita Vasilev)
- Increase OOM score in debug version on Linux. #6152 (akuzm)
- HDFS HA now work in debug build. #6650 (Weiqing Xu)
- Added a test to transform\_query\_for\_external\_database. #6388 (alexey-milovidov)
- Add test for multiple materialized views for Kafka table. #6509 (Ivan)
- Make a better build scheme. #6500 (Ivan)
- Fixed test\_external\_dictionaries integration in case it was executed under non root user. #6507 (Nikolai Kochetov)
- The bug reproduces when total size of written packets exceeds DBMS\_DEFAULT\_BUFFER\_SIZE. #6204 (Yuriy Baranov)
- Added a test for RENAME table race condition #6752 (alexey-milovidov)
- Avoid data race on Settings in KILL QUERY. #6753 (alexey-milovidov)
- Add integration test for handling errors by a cache dictionary. #6755 (Vitaly Baranov)
- Disable parsing of ELF object files on Mac OS, because it makes no sense. #6578 (alexey-milovidov)

- Attempt to make changelog generator better. #6327 (alexey-milovidov)
- Adding `-Wshadow` switch to the GCC. #6325 (kreuzerkrieg)
- Removed obsolete code for `mimalloc` support. #6715 (alexey-milovidov)
- `zlib-ng` determines x86 capabilities and saves this info to global variables. This is done in `defalteInit` call, which may be made by different threads simultaneously. To avoid multithreaded writes, do it on library startup. #6141 (akuzm)
- Regression test for a bug which in `join` which was fixed in #5192. #6147 (Bakhtiyor Ruziev)
- Fixed MSan report. #6144 (alexey-milovidov)
- Fix flapping TTL test. #6782 (Anton Popov)
- Fixed false data race in `MergeTreeDataPart::is_frozen` field. #6583 (alexey-milovidov)
- Fixed timeouts in fuzz test. In previous version, it managed to find false hangup in query `SELECT * FROM numbers_mt(gccMurmurHash(""))`. #6582 (alexey-milovidov)
- Added debug checks to `static_cast` of columns. #6581 (alexey-milovidov)
- Support for Oracle Linux in official RPM packages. #6356 #6585 (alexey-milovidov)
- Changed json perftests from `once` to `loop` type. #6536 (Nikolai Kochetov)
- `odbc-bridge.cpp` defines `main()` so it should not be included in `clickhouse-lib`. #6538 (Orivej Desh)
- Test for crash in `FULL|RIGHT JOIN` with nulls in right table's keys. #6362 (Artem Zuikov)
- Added a test for the limit on expansion of aliases just in case. #6442 (alexey-milovidov)
- Switched from `boost::filesystem` to `std::filesystem` where appropriate. #6253 #6385 (alexey-milovidov)
- Added RPM packages to website. #6251 (alexey-milovidov)
- Add a test for fixed `Unknown identifier` exception in `IN` section. #6708 (Artem Zuikov)
- Simplify `shared_ptr_helper` because people facing difficulties understanding it. #6675 (alexey-milovidov)
- Added performance tests for fixed Gorilla and DoubleDelta codec. #6179 (Vasily Nemkov)
- Split the integration test `test_dictionaries` into 4 separate tests. #6776 (Vitaly Baranov)
- Fix PVS-Studio warning in `PipelineExecutor`. #6777 (Nikolai Kochetov)
- Allow to use `library` dictionary source with ASan. #6482 (alexey-milovidov)
- Added option to generate changelog from a list of PRs. #6350 (alexey-milovidov)
- Lock the `TinyLog` storage when reading. #6226 (akuzm)
- Check for broken symlinks in CI. #6634 (alexey-milovidov)
- Increase timeout for “stack overflow” test because it may take a long time in debug build. #6637 (alexey-milovidov)
- Added a check for double whitespaces. #6643 (alexey-milovidov)
- Fix `new/delete` memory tracking when build with sanitizers. Tracking is not clear. It only prevents memory limit exceptions in tests. #6450 (Artem Zuikov)
- Enable back the check of undefined symbols while linking. #6453 (Ivan)

- Avoid rebuilding `hyperscan` every day. [#6307 \(alexey-milovidov\)](#)
- Fixed UBSan report in `ProtobufWriter`. [#6163 \(alexey-milovidov\)](#)
- Don't allow to use query profiler with sanitizers because it is not compatible. [#6769 \(alexey-milovidov\)](#)
- Add test for reloading a dictionary after fail by timer. [#6114 \(Vitaly Baranov\)](#)
- Fix inconsistency in `PipelineExecutor::prepareProcessor` argument type. [#6494 \(Nikolai Kochetov\)](#)
- Added a test for bad URLs. [#6493 \(alexey-milovidov\)](#)
- Added more checks to `CAST` function. This should get more information about segmentation fault in fuzzy test. [#6346 \(Nikolai Kochetov\)](#)
- Added `gcc-9` support to `docker/builder` container that builds image locally. [#6333 \(Gleb Novikov\)](#)
- Test for primary key with `LowCardinality(String)`. [#5044 #6219 \(dimarub2000\)](#)
- Fixed tests affected by slow stack traces printing. [#6315 \(alexey-milovidov\)](#)
- Add a test case for crash in `groupUniqArray` fixed in [#6029. #4402 #6129 \(akuzm\)](#)
- Fixed indices mutations tests. [#6645 \(Nikita Vasilev\)](#)
- In performance test, do not read query log for queries we didn't run. [#6427 \(akuzm\)](#)
- Materialized view now could be created with any low cardinality types regardless to the setting about suspicious low cardinality types. [#6428 \(Olga Khvostikova\)](#)
- Updated tests for `send_logs_level` setting. [#6207 \(Nikolai Kochetov\)](#)
- Fix build under gcc-8.2. [#6196 \(Max Akhmedov\)](#)
- Fix build with internal libc++. [#6724 \(Ivan\)](#)
- Fix shared build with `rdkafka` library [#6101 \(Ivan\)](#)
- Fixes for Mac OS build (incomplete). [#6390 \(alexey-milovidov\) #6429 \(alex-zaitsev\)](#)
- Fix "splitted" build. [#6618 \(alexey-milovidov\)](#)
- Other build fixes: [#6186 \(Amos Bird\) #6486 #6348 \(vxider\) #6744 \(Ivan\) #6016 #6421 #6491 \(proller\)](#)

## Backward Incompatible Change

- Removed rarely used table function `catBoostPool` and storage `CatBoostPool`. If you have used this table function, please write email to `clickhouse-feedback@yandex-team.com`. Note that CatBoost integration remains and will be supported. [#6279 \(alexey-milovidov\)](#)
- Disable `ANY RIGHT JOIN` and `ANY FULL JOIN` by default. Set `any_join_distinct_right_table_keys` setting to enable them. [#5126 #6351 \(Artem Zuikov\)](#)

## ClickHouse Release 19.13

### ClickHouse Release 19.13.6.51, 2019-10-02

#### Bug Fix

- This release also contains all bug fixes from 19.11.12.69.

### ClickHouse Release 19.13.5.44, 2019-09-20

#### Bug Fix

- This release also contains all bug fixes from 19.14.6.12.
- Fixed possible inconsistent state of table while executing `DROP` query for replicated table while zookeeper is not accessible. [#6045 #6413](#) ([Nikita Mikhaylov](#))
- Fix for data race in StorageMerge [#6717](#) ([alexey-milovidov](#))
- Fix bug introduced in query profiler which leads to endless recv from socket. [#6386](#) ([alesapin](#))
- Fix excessive CPU usage while executing `JSONExtractRaw` function over a boolean value. [#6208](#) ([Vitaly Baranov](#))
- Fixes the regression while pushing to materialized view. [#6415](#) ([Ivan](#))
- Table function `url` had the vulnerability allowed the attacker to inject arbitrary HTTP headers in the request. This issue was found by [Nikita Tikhomirov](#). [#6466](#) ([alexey-milovidov](#))
- Fix useless `AST` check in Set index. [#6510 #6651](#) ([Nikita Vasilev](#))
- Fixed parsing of `AggregateFunction` values embedded in query. [#6575 #6773](#) ([Zhichang Yu](#))
- Fixed wrong behaviour of `trim` functions family. [#6647](#) ([alexey-milovidov](#))

## ClickHouse Release 19.13.4.32, 2019-09-10

### Bug Fix

- This release also contains all bug security fixes from 19.11.9.52 and 19.11.10.54.
- Fixed data race in `system.parts` table and `ALTER` query. [#6245 #6513](#) ([alexey-milovidov](#))
- Fixed mismatched header in streams happened in case of reading from empty distributed table with sample and prewhere. [#6167](#) ([Lixiang Qian](#)) [#6823](#) ([Nikolai Kochetov](#))
- Fixed crash when using `IN` clause with a subquery with a tuple. [#6125 #6550](#) ([tavplubix](#))
- Fix case with same column names in `GLOBAL JOIN ON` section. [#6181](#) ([Artem Zuikov](#))
- Fix crash when casting types to `Decimal` that do not support it. Throw exception instead. [#6297](#) ([Artem Zuikov](#))
- Fixed crash in `extractAll()` function. [#6644](#) ([Artem Zuikov](#))
- Query transformation for MySQL, ODBC, JDBC table functions now works properly for `SELECT WHERE` queries with multiple `AND` expressions. [#6381 #6676](#) ([dimarub2000](#))
- Added previous declaration checks for MySQL 8 integration. [#6569](#) ([Rafael David Tinoco](#))

### Security Fix

- Fix two vulnerabilities in codecs in decompression phase (malicious user can fabricate compressed data that will lead to buffer overflow in decompression). [#6670](#) ([Artem Zuikov](#))

## ClickHouse Release 19.13.3.26, 2019-08-22

### Bug Fix

- Fix `ALTER TABLE ... UPDATE` query for tables with `enable_mixed_granularity_parts=1`. [#6543](#) ([alesapin](#))
- Fix NPE when using `IN` clause with a subquery with a tuple. [#6125 #6550](#) ([tavplubix](#))
- Fixed an issue that if a stale replica becomes alive, it may still have data parts that were removed by `DROP PARTITION`. [#6522 #6523](#) ([tavplubix](#))

- Fixed issue with parsing CSV [#6426](#) [#6559](#) ([tavplubix](#))
- Fixed data race in system.parts table and ALTER query. This fixes [#6245](#), [#6513](#) ([alexey-milovidov](#))
- Fixed wrong code in mutations that may lead to memory corruption. Fixed segfault with read of address `0x14c0` that may happen due to concurrent `DROP TABLE` and `SELECT` from `system.parts` or `system.parts_columns`. Fixed race condition in preparation of mutation queries. Fixed deadlock caused by `OPTIMIZE` of Replicated tables and concurrent modification operations like ALTERs. [#6514](#) ([alexey-milovidov](#))
- Fixed possible data loss after `ALTER DELETE` query on table with skipping index. [#6224](#) [#6282](#) ([Nikita Vasilev](#))

## Security Fix

- If the attacker has write access to ZooKeeper and is able to run custom server available from the network where ClickHouse runs, it can create custom-built malicious server that will act as ClickHouse replica and register it in ZooKeeper. When another replica will fetch data part from malicious replica, it can force clickhouse-server to write to arbitrary path on filesystem. Found by Eldar Zaitov, information security team at Yandex. [#6247](#) ([alexey-milovidov](#))

## ClickHouse Release 19.13.2.19, 2019-08-14

### New Feature

- Sampling profiler on query level. Example. [#4247](#) ([laplab](#)) [#6124](#) ([alexey-milovidov](#)) [#6250](#) [#6283](#) [#6386](#)
- Allow to specify a list of columns with `COLUMNS('regexp')` expression that works like a more sophisticated variant of `*` asterisk. [#5951](#) ([mfridental](#)), ([alexey-milovidov](#))
- `CREATE TABLE AS table_function()` is now possible [#6057](#) ([dimarub2000](#))
- Adam optimizer for stochastic gradient descent is used by default in `stochasticLinearRegression()` and `stochasticLogisticRegression()` aggregate functions, because it shows good quality without almost any tuning. [#6000](#) ([Quid37](#))
- Added functions for working with the custom week number [#5212](#) ([Andy Yang](#))
- `RENAME` queries now work with all storages. [#5953](#) ([Ivan](#))
- Now client receive logs from server with any desired level by setting `send_logs_level` regardless to the log level specified in server settings. [#5964](#) ([Nikita Mikhaylov](#))

### Backward Incompatible Change

- The setting `input_format_defaults_for_omitted_fields` is enabled by default. Inserts in Distributed tables need this setting to be the same on cluster (you need to set it before rolling update). It enables calculation of complex default expressions for omitted fields in `JSONEachRow` and `CSV*` formats. It should be the expected behavior but may lead to negligible performance difference. [#6043](#) ([Artem Zuikov](#)), [#5625](#) ([akuzm](#))

### Experimental Features

- New query processing pipeline. Use `experimental_use_processors=1` option to enable it. Use for your own trouble. [#4914](#) ([Nikolai Kochetov](#))

### Bug Fix

- Kafka integration has been fixed in this version.

- Fixed DoubleDelta encoding of Int64 for large DoubleDelta values, improved DoubleDelta encoding for random data for Int32. [#5998](#) ([Vasily Nemkov](#))
- Fixed overestimation of max\_rows\_to\_read if the setting merge\_tree\_uniform\_read\_distribution is set to 0. [#6019](#) ([alexey-milovidov](#))

## Improvement

- Throws an exception if config.d file does not have the corresponding root element as the config file [#6123](#) ([dimarub2000](#))

## Performance Improvement

- Optimize count(). Now it uses the smallest column (if possible). [#6028](#) ([Amos Bird](#))

## Build/Testing/Packaging Improvement

- Report memory usage in performance tests. [#5899](#) ([akuzm](#))
- Fix build with external libcxx [#6010](#) ([Ivan](#))
- Fix shared build with rdkafka library [#6101](#) ([Ivan](#))

# ClickHouse Release 19.11

## ClickHouse Release 19.11.13.74, 2019-11-01

### Bug Fix

- Fixed rare crash in ALTER MODIFY COLUMN and vertical merge when one of merged/altered parts is empty (0 rows). [#6780](#) ([alesapin](#))
- Manual update of SIMDJSON. This fixes possible flooding of stderr files with bogus json diagnostic messages. [#7548](#) ([Alexander Kazakov](#))
- Fixed bug with mrk file extension for mutations ([alesapin](#))

## ClickHouse Release 19.11.12.69, 2019-10-02

### Bug Fix

- Fixed performance degradation of index analysis on complex keys on large tables. This fixes [#6924](#). [#7075](#) ([alexey-milovidov](#))
- Avoid rare SIGSEGV while sending data in tables with Distributed engine (Failed to send batch: file with index XXXXX is absent). [#7032](#) ([Azat Khuzhin](#))
- Fix Unknown identifier with multiple joins. This fixes [#5254](#). [#7022](#) ([Artem Zuikov](#))

## ClickHouse Release 19.11.11.57, 2019-09-13

- Fix logical error causing segfaults when selecting from Kafka empty topic. [#6902](#) [#6909](#) ([Ivan](#))
- Fix for function ArrayEnumerateUniqRanked with empty arrays in params. [#6928](#) ([proller](#))

## ClickHouse Release 19.11.10.54, 2019-09-10

### Bug Fix

- Do store offsets for Kafka messages manually to be able to commit them all at once for all partitions. Fixes potential duplication in “one consumer - many partitions” scenario. [#6872](#) ([Ivan](#))

## ClickHouse Release 19.11.9.52, 2019-09-06

- Improve error handling in cache dictionaries. #6737 (Vitaly Baranov)
- Fixed bug in function `arrayEnumerateUniqRanked`. #6779 (proller)
- Fix `JSONExtract` function while extracting a `Tuple` from JSON. #6718 (Vitaly Baranov)
- Fixed possible data loss after `ALTER DELETE` query on table with skipping index. #6224 #6282 (Nikita Vasilev)
- Fixed performance test. #6392 (alexey-milovidov)
- Parquet: Fix reading boolean columns. #6579 (alexey-milovidov)
- Fixed wrong behaviour of `nullIf` function for constant arguments. #6518 (Guillaume Tassery) #6580 (alexey-milovidov)
- Fix Kafka messages duplication problem on normal server restart. #6597 (Ivan)
- Fixed an issue when long `ALTER UPDATE` or `ALTER DELETE` may prevent regular merges to run. Prevent mutations from executing if there is no enough free threads available. #6502 #6617 (tavplubix)
- Fixed error with processing “timezone” in server configuration file. #6709 (alexey-milovidov)
- Fix kafka tests. #6805 (Ivan)

## Security Fix

- If the attacker has write access to ZooKeeper and is able to run custom server available from the network where ClickHouse runs, it can create custom-built malicious server that will act as ClickHouse replica and register it in ZooKeeper. When another replica will fetch data part from malicious replica, it can force clickhouse-server to write to arbitrary path on filesystem. Found by Eldar Zaitov, information security team at Yandex. #6247 (alexey-milovidov)

## ClickHouse Release 19.11.8.46, 2019-08-22

### Bug Fix

- Fix `ALTER TABLE ... UPDATE` query for tables with `enable_mixed_granularity_parts=1`. #6543 (alesapin)
- Fix NPE when using `IN` clause with a subquery with a tuple. #6125 #6550 (tavplubix)
- Fixed an issue that if a stale replica becomes alive, it may still have data parts that were removed by `DROP PARTITION`. #6522 #6523 (tavplubix)
- Fixed issue with parsing CSV #6426 #6559 (tavplubix)
- Fixed data race in `system.parts` table and `ALTER` query. This fixes #6245. #6513 (alexey-milovidov)
- Fixed wrong code in mutations that may lead to memory corruption. Fixed segfault with read of address `0x14c0` that may happen due to concurrent `DROP TABLE` and `SELECT` from `system.parts` or `system.parts_columns`. Fixed race condition in preparation of mutation queries. Fixed deadlock caused by `OPTIMIZE` of Replicated tables and concurrent modification operations like `ALTERs`. #6514 (alexey-milovidov)

## ClickHouse Release 19.11.7.40, 2019-08-14

### Bug Fix

- Kafka integration has been fixed in this version.
- Fix segfault when using `arrayReduce` for constant arguments. #6326 (alexey-milovidov)
- Fixed `toFloat()` monotonicity. #6374 (dimarub2000)

- Fix segfault with enabled `optimize_skip_unused_shards` and missing sharding key. #6384 (Curtiz)
- Fixed logic of `arrayEnumerateUniqRanked` function. #6423 (alexey-milovidov)
- Removed extra verbose logging from MySQL handler. #6389 (alexey-milovidov)
- Fix wrong behavior and possible segfaults in `topK` and `topKWeighted` aggregated functions. #6404 (Curtiz)
- Do not expose virtual columns in `system.columns` table. This is required for backward compatibility. #6406 (alexey-milovidov)
- Fix bug with memory allocation for string fields in complex key cache dictionary. #6447 (alesapin)
- Fix bug with enabling adaptive granularity when creating new replica for `Replicated*MergeTree` table. #6452 (alesapin)
- Fix infinite loop when reading Kafka messages. #6354 (abyss7)
- Fixed the possibility of a fabricated query to cause server crash due to stack overflow in SQL parser and possibility of stack overflow in `Merge` and `Distributed` tables #6433 (alexey-milovidov)
- Fixed Gorilla encoding error on small sequences. #6444 (Enmk)

## Improvement

- Allow user to override `poll_interval` and `idle_connection_timeout` settings on connection. #6230 (alexey-milovidov)

## ClickHouse Release 19.11.5.28, 2019-08-05

### Bug Fix

- Fixed the possibility of hanging queries when server is overloaded. #6301 (alexey-milovidov)
- Fix FPE in `yandexConsistentHash` function. This fixes #6304. #6126 (alexey-milovidov)
- Fixed bug in conversion of `LowCardinality` types in `AggregateFunctionFactory`. This fixes #6257. #6281 (Nikolai Kochetov)
- Fix parsing of `bool` settings from `true` and `false` strings in configuration files. #6278 (alesapin)
- Fix rare bug with incompatible stream headers in queries to `Distributed` table over `MergeTree` table when part of `WHERE` moves to `PREWHERE`. #6236 (alesapin)
- Fixed overflow in integer division of signed type to unsigned type. This fixes #6214. #6233 (alexey-milovidov)

### Backward Incompatible Change

- Kafka still broken.

## ClickHouse Release 19.11.4.24, 2019-08-01

### Bug Fix

- Fix bug with writing secondary indices marks with adaptive granularity. #6126 (alesapin)
- Fix `WITH ROLLUP` and `WITH CUBE` modifiers of `GROUP BY` with two-level aggregation. #6225 (Anton Popov)
- Fixed hang in `JSONExtractRaw` function. Fixed #6195 #6198 (alexey-milovidov)
- Fix segfault in `ExternalLoader::reloadOutdated()`. #6082 (Vitaly Baranov)

- Fixed the case when server may close listening sockets but not shutdown and continue serving remaining queries. You may end up with two running clickhouse-server processes. Sometimes, the server may return an error `bad_function_call` for remaining queries. [#6231 \(alexey-milovidov\)](#)
- Fixed useless and incorrect condition on update field for initial loading of external dictionaries via ODBC, MySQL, ClickHouse and HTTP. This fixes [#6069 #6083 \(alexey-milovidov\)](#)
- Fixed irrelevant exception in cast of `LowCardinality(Nullable)` to not-Nullable column in case if it does not contain Nulls (e.g. in query like `SELECT CAST(CAST('Hello' AS LowCardinality(Nullable(String))) AS String)`). [#6094 #6119 \(Nikolai Kochetov\)](#)
- Fix non-deterministic result of “`uniq`” aggregate function in extreme rare cases. The bug was present in all ClickHouse versions. [#6058 \(alexey-milovidov\)](#)
- Segfault when we set a little bit too high CIDR on the function `IPv6CIDRToRange`. [#6068 \(Guillaume Tassery\)](#)
- Fixed small memory leak when server throw many exceptions from many different contexts. [#6144 \(alexey-milovidov\)](#)
- Fix the situation when consumer got paused before subscription and not resumed afterwards. [#6075 \(Ivan\)](#) Note that Kafka is broken in this version.
- Clearing the Kafka data buffer from the previous read operation that was completed with an error [#6026 \(Nikolay\)](#) Note that Kafka is broken in this version.
- Since `StorageMergeTree::background_task_handle` is initialized in `startup()` the `MergeTreeBlockOutputStream::write()` may try to use it before initialization. Just check if it is initialized. [#6080 \(Ivan\)](#)

## Build/Testing/Packaging Improvement

- Added official `rpm` packages. [#5740 \(proller\) \(alesapin\)](#)
- Add an ability to build `.rpm` and `.tgz` packages with `packager` script. [#5769 \(alesapin\)](#)
- Fixes for “Arcadia” build system. [#6223 \(proller\)](#)

## Backward Incompatible Change

- Kafka is broken in this version.

# ClickHouse Release 19.11.3.11, 2019-07-18

## New Feature

- Added support for prepared statements. [#5331 \(Alexander\) #5630 \(alexey-milovidov\)](#)
- `DoubleDelta` and `Gorilla` column codecs [#5600 \(Vasily Nemkov\)](#)
- Added `os_thread_priority` setting that allows to control the “nice” value of query processing threads that is used by OS to adjust dynamic scheduling priority. It requires `CAP_SYS_NICE` capabilities to work. This implements [#5858 #5909 \(alexey-milovidov\)](#)
- Implement `_topic`, `_offset`, `_key` columns for Kafka engine [#5382 \(Ivan\)](#) Note that Kafka is broken in this version.
- Add aggregate function combinator `-Resample` [#5590 \(hcz\)](#)
- Aggregate functions `groupArrayMovingSum(win_size)(x)` and `groupArrayMovingAvg(win_size)(x)`, which calculate moving sum/avg with or without window-size limitation. [#5595 \(inv2004\)](#)

- Add synonym `arrayFlatten \<-> flatten` #5764 (hcz)
- Integrate H3 function `geoToH3` from Uber. #4724 (Remen Ivan) #5805 (alexey-milovidov)

## Bug Fix

- Implement DNS cache with asynchronous update. Separate thread resolves all hosts and updates DNS cache with period (setting `dns_cache_update_period`). It should help, when ip of hosts changes frequently. #5857 (Anton Popov)
- Fix segfault in `Delta` codec which affects columns with values less than 32 bits size. The bug led to random memory corruption. #5786 (alesapin)
- Fix segfault in TTL merge with non-physical columns in block. #5819 (Anton Popov)
- Fix rare bug in checking of part with `LowCardinality` column. Previously `checkDataPart` always fails for part with `LowCardinality` column. #5832 (alesapin)
- Avoid hanging connections when server thread pool is full. It is important for connections from `remote` table function or connections to a shard without replicas when there is long connection timeout. This fixes #5878 #5881 (alexey-milovidov)
- Support for constant arguments to `evalMLModel` function. This fixes #5817 #5820 (alexey-milovidov)
- Fixed the issue when ClickHouse determines default time zone as `UCT` instead of `UTC`. This fixes #5804. #5828 (alexey-milovidov)
- Fixed buffer underflow in `visitParamExtractRaw`. This fixes #5901 #5902 (alexey-milovidov)
- Now distributed `DROP/ALTER/TRUNCATE/OPTIMIZE ON CLUSTER` queries will be executed directly on leader replica. #5757 (alesapin)
- Fix coalesce for `ColumnConst` with `ColumnNullable` + related changes. #5755 (Artem Zuikov)
- Fix the `ReadBufferFromKafkaConsumer` so that it keeps reading new messages after `commit()` even if it was stalled before #5852 (Ivan)
- Fix `FULL` and `RIGHT JOIN` results when joining on `Nullable` keys in right table. #5859 (Artem Zuikov)
- Possible fix of infinite sleeping of low-priority queries. #5842 (alexey-milovidov)
- Fix race condition, which cause that some queries may not appear in `query_log` after `SYSTEM FLUSH LOGS` query. #5456 #5685 (Anton Popov)
- Fixed `heap-use-after-free` ASan warning in `ClusterCopier` caused by watch which try to use already removed copier object. #5871 (Nikolai Kochetov)
- Fixed wrong `StringRef` pointer returned by some implementations of `IColumn::deserializeAndInsertFromArena`. This bug affected only unit-tests. #5973 (Nikolai Kochetov)
- Prevent source and intermediate array join columns of masking same name columns. #5941 (Artem Zuikov)
- Fix insert and select query to MySQL engine with MySQL style identifier quoting. #5704 (Winter Zhang)
- Now `CHECK TABLE` query can work with MergeTree engine family. It returns check status and message if any for each part (or file in case of simpler engines). Also, fix bug in fetch of a broken part. #5865 (alesapin)
- Fix `SPLIT_SHARED_LIBRARIES` runtime #5793 (Danila Kutenin)

- Fixed time zone initialization when `/etc/localtime` is a relative symlink like `../usr/share/zoneinfo/Europe/Moscow` [#5922 \(alexey-milovidov\)](#)
- clickhouse-copier: Fix use-after free on shutdown [#5752 \(proller\)](#)
- Updated `simdjson`. Fixed the issue that some invalid JSONs with zero bytes successfully parse. [#5938 \(alexey-milovidov\)](#)
- Fix shutdown of SystemLogs [#5802 \(Anton Popov\)](#)
- Fix hanging when condition in `invalidate_query` depends on a dictionary. [#6011 \(Vitaly Baranov\)](#)

## Improvement

- Allow unresolvable addresses in cluster configuration. They will be considered unavailable and tried to resolve at every connection attempt. This is especially useful for Kubernetes. This fixes [#5714](#) [#5924 \(alexey-milovidov\)](#)
- Close idle TCP connections (with one hour timeout by default). This is especially important for large clusters with multiple distributed tables on every server, because every server can possibly keep a connection pool to every other server, and after peak query concurrency, connections will stall. This fixes [#5879](#) [#5880 \(alexey-milovidov\)](#)
- Better quality of `topK` function. Changed the `SavingSpace` set behavior to remove the last element if the new element have a bigger weight. [#5833](#) [#5850 \(Guillaume Tassery\)](#)
- URL functions to work with domains now can work for incomplete URLs without scheme [#5725 \(alesapin\)](#)
- Checksums added to the `system.parts_columns` table. [#5874 \(Nikita Mikhaylov\)](#)
- Added `Enum` data type as a synonym for `Enum8` or `Enum16`. [#5886 \(dimarub2000\)](#)
- Full bit transpose variant for `T64` codec. Could lead to better compression with `zstd`. [#5742 \(Artem Zuikov\)](#)
- Condition on `startsWith` function now can uses primary key. This fixes [#5310](#) and [#5882](#) [#5919 \(dimarub2000\)](#)
- Allow to use `clickhouse-copier` with cross-replication cluster topology by permitting empty database name. [#5745 \(nvartolomei\)](#)
- Use `UTC` as default timezone on a system without `tzdata` (e.g. bare Docker container). Before this patch, error message `Could not determine local time zone` was printed and server or client refused to start. [#5827 \(alexey-milovidov\)](#)
- Returned back support for floating point argument in function `quantileTiming` for backward compatibility. [#5911 \(alexey-milovidov\)](#)
- Show which table is missing column in error messages. [#5768 \(Ivan\)](#)
- Disallow run query with same `query_id` by various users [#5430 \(proller\)](#)
- More robust code for sending metrics to Graphite. It will work even during long multiple `RENAME TABLE` operation. [#5875 \(alexey-milovidov\)](#)
- More informative error messages will be displayed when ThreadPool cannot schedule a task for execution. This fixes [#5305](#) [#5801 \(alexey-milovidov\)](#)
- Inverting ngramSearch to be more intuitive [#5807 \(Danila Kutenin\)](#)
- Add user parsing in HDFS engine builder [#5946 \(akonyaev90\)](#)

- Update default value of `max_ast_elements` parameter [#5933](#) ([Artem Konovalov](#))
- Added a notion of obsolete settings. The obsolete setting `allow_experimental_low_cardinality_type` can be used with no effect. [0f15c01c6802f7ce1a1494c12c846be8c98944cd](#) [Alexey Milovidov](#)

## Performance Improvement

- Increase number of streams to SELECT from Merge table for more uniform distribution of threads. Added setting `max_streams_multiplier_for_merge_tables`. This fixes [#5797](#) [#5915](#) ([alexey-milovidov](#))

## Build/Testing/Packaging Improvement

- Add a backward compatibility test for client-server interaction with different versions of clickhouse. [#5868](#) ([alesapin](#))
- Test coverage information in every commit and pull request. [#5896](#) ([alesapin](#))
- Cooperate with address sanitizer to support our custom allocators (Arena and ArenaWithFreeLists) for better debugging of “use-after-free” errors. [#5728](#) ([akuzm](#))
- Switch to [LLVM libunwind implementation](#) for C++ exception handling and for stack traces printing [#4828](#) ([Nikita Lapkov](#))
- Add two more warnings from -Weverything [#5923](#) ([alexey-milovidov](#))
- Allow to build ClickHouse with Memory Sanitizer. [#3949](#) ([alexey-milovidov](#))
- Fixed ubsan report about `bitTest` function in fuzz test. [#5943](#) ([alexey-milovidov](#))
- Docker: added possibility to init a ClickHouse instance which requires authentication. [#5727](#) ([Korviakov Andrey](#))
- Update librdkafka to version 1.1.0 [#5872](#) ([Ivan](#))
- Add global timeout for integration tests and disable some of them in tests code. [#5741](#) ([alesapin](#))
- Fix some ThreadSanitizer failures. [#5854](#) ([akuzm](#))
- The `--no-undefined` option forces the linker to check all external names for existence while linking. It's very useful to track real dependencies between libraries in the split build mode. [#5855](#) ([Ivan](#))
- Added performance test for [#5797](#) [#5914](#) ([alexey-milovidov](#))
- Fixed compatibility with gcc-7. [#5840](#) ([alexey-milovidov](#))
- Added support for gcc-9. This fixes [#5717](#) [#5774](#) ([alexey-milovidov](#))
- Fixed error when libunwind can be linked incorrectly. [#5948](#) ([alexey-milovidov](#))
- Fixed a few warnings found by PVS-Studio. [#5921](#) ([alexey-milovidov](#))
- Added initial support for `clang-tidy` static analyzer. [#5806](#) ([alexey-milovidov](#))
- Convert BSD/Linux endian macros( ‘be64toh’ and ‘htobe64’) to the Mac OS X equivalents [#5785](#) ([Fu Chen](#))
- Improved integration tests guide. [#5796](#) ([Vladimir Chebotarev](#))
- Fixing build at macosx + gcc9 [#5822](#) ([filimonov](#))
- Fix a hard-to-spot typo: aggreAGte -> aggregate. [#5753](#) ([akuzm](#))
- Fix freebsd build [#5760](#) ([proller](#))

- Add link to experimental YouTube channel to website #5845 (Ivan Blinkov)
- CMake: add option for coverage flags: WITH\_COVERAGE #5776 (proller)
- Fix initial size of some inline PODArray's. #5787 (akuzm)
- clickhouse-server.postinst: fix os detection for centos 6 #5788 (proller)
- Added Arch linux package generation. #5719 (Vladimir Chebotarev)
- Split Common/config.h by libs (dbms) #5715 (proller)
- Fixes for “Arcadia” build platform #5795 (proller)
- Fixes for unconventional build (gcc9, no submodules) #5792 (proller)
- Require explicit type in unalignedStore because it was proven to be bug-prone #5791 (akuzm)
- Fixes MacOS build #5830 (filimonov)
- Performance test concerning the new JIT feature with bigger dataset, as requested here #5263 #5887 (Guillaume Tassery)
- Run stateful tests in stress test 12693e568722f11e19859742f56428455501fd2a (alesapin)

## Backward Incompatible Change

- Kafka is broken in this version.
- Enable adaptive\_index\_granularity = 10MB by default for new MergeTree tables. If you created new MergeTree tables on version 19.11+, downgrade to versions prior to 19.6 will be impossible. #5628 (alesapin)
- Removed obsolete undocumented embedded dictionaries that were used by Yandex.Metrica. The functions OSIn, SEIn, OSToRoot, SEToRoot, OSHierarchy, SEHierarchy are no longer available. If you are using these functions, write email to [clickhouse-feedback@yandex-team.com](mailto:clickhouse-feedback@yandex-team.com). Note: at the last moment we decided to keep these functions for a while. #5780 (alexey-milovidov)

# ClickHouse Release 19.10

## ClickHouse Release 19.10.1.5, 2019-07-12

### New Feature

- Add new column codec: T64. Made for (U)IntX/EnumX/Data(Time)/DecimalX columns. It should be good for columns with constant or small range values. Codec itself allows enlarge or shrink data type without re-compression. #5557 (Artem Zuikov)
- Add database engine MySQL that allow to view all the tables in remote MySQL server #5599 (Winter Zhang)
- bitmapContains implementation. It's 2x faster than bitmapHasAny if the second bitmap contains one element. #5535 (Zhichang Yu)
- Support for crc32 function (with behaviour exactly as in MySQL or PHP). Do not use it if you need a hash function. #5661 (Remen Ivan)
- Implemented SYSTEM START/STOP DISTRIBUTED SENDS queries to control asynchronous inserts into Distributed tables. #4935 (Winter Zhang)

### Bug Fix

- Ignore query execution limits and max parts size for merge limits while executing mutations. #5659 (Anton Popov)
- Fix bug which may lead to deduplication of normal blocks (extremely rare) and insertion of duplicate blocks (more often). #5549 (alesapin)
- Fix of function `arrayEnumerateUniqRanked` for arguments with empty arrays #5559 (proller)
- Don't subscribe to Kafka topics without intent to poll any messages. #5698 (Ivan)
- Make setting `join_use_nulls` get no effect for types that cannot be inside Nullable #5700 (Olga Khvostikova)
- Fixed `Incorrect size of index granularity` errors #5720 (coraxster)
- Fix Float to Decimal convert overflow #5607 (coraxster)
- Flush buffer when `WriteBufferFromHDFS`'s destructor is called. This fixes writing into HDFS. #5684 (Xindong Peng)

## Improvement

- Treat empty cells in `csv` as default values when the setting `input_format_defaults_for_omitted_fields` is enabled. #5625 (akuzm)
- Non-blocking loading of external dictionaries. #5567 (Vitaly Baranov)
- Network timeouts can be dynamically changed for already established connections according to the settings. #4558 (Konstantin Podshumok)
- Using “`public_suffix_list`” for functions `firstSignificantSubdomain`, `cutToFirstSignificantSubdomain`. It's using a perfect hash table generated by `gperf` with a list generated from the file: [https://publicsuffix.org/list/public\\_suffix\\_list.dat](https://publicsuffix.org/list/public_suffix_list.dat). (for example, now we recognize the domain ac.uk as non-significant). #5030 (Guillaume Tassery)
- Adopted `IPv6` data type in system tables; unified client info columns in `system.processes` and `system.query_log` #5640 (alexey-milovidov)
- Using sessions for connections with MySQL compatibility protocol. #5476 #5646 (Yuriy Baranov)
- Support more `ALTER` queries `ON CLUSTER`. #5593 #5613 (sundyli)
- Support `<logger>` section in `clickhouse-local` config file. #5540 (proller)
- Allow run query with `remote` table function in `clickhouse-local` #5627 (proller)

## Performance Improvement

- Add the possibility to write the final mark at the end of MergeTree columns. It allows to avoid useless reads for keys that are out of table data range. It is enabled only if adaptive index granularity is in use. #5624 (alesapin)
- Improved performance of MergeTree tables on very slow filesystems by reducing number of `stat` syscalls. #5648 (alexey-milovidov)
- Fixed performance degradation in reading from MergeTree tables that was introduced in version 19.6. Fixes #5631. #5633 (alexey-milovidov)

## Build/Testing/Packaging Improvement

- Implemented `TestKeeper` as an implementation of ZooKeeper interface used for testing #5643 (alexey-milovidov) (levushkin aleksej)

- From now on `.sql` tests can be run isolated by server, in parallel, with random database. It allows to run them faster, add new tests with custom server configurations, and be sure that different tests does not affect each other. [#5554](#) ([Ivan](#))
- Remove `<name>` and `<metrics>` from performance tests [#5672](#) ([Olga Khvostikova](#))
- Fixed “select\_format” performance test for `Pretty` formats [#5642](#) ([alexey-milovidov](#))

## ClickHouse Release 19.9

### ClickHouse Release 19.9.3.31, 2019-07-05

#### Bug Fix

- Fix segfault in Delta codec which affects columns with values less than 32 bits size. The bug led to random memory corruption. [#5786](#) ([alesapin](#))
- Fix rare bug in checking of part with LowCardinality column. [#5832](#) ([alesapin](#))
- Fix segfault in TTL merge with non-physical columns in block. [#5819](#) ([Anton Popov](#))
- Fix potential infinite sleeping of low-priority queries. [#5842](#) ([alexey-milovidov](#))
- Fix how ClickHouse determines default time zone as UCT instead of UTC. [#5828](#) ([alexey-milovidov](#))
- Fix bug about executing distributed DROP/ALTER/TRUNCATE/OPTIMIZE ON CLUSTER queries on follower replica before leader replica. Now they will be executed directly on leader replica. [#5757](#) ([alesapin](#))
- Fix race condition, which cause that some queries may not appear in `query_log` instantly after `SYSTEM FLUSH LOGS` query. [#5685](#) ([Anton Popov](#))
- Added missing support for constant arguments to `evalMLModel` function. [#5820](#) ([alexey-milovidov](#))

### ClickHouse Release 19.9.2.4, 2019-06-24

#### New Feature

- Print information about frozen parts in `system.parts` table. [#5471](#) ([proller](#))
- Ask client password on `clickhouse-client` start on `tty` if not set in arguments [#5092](#) ([proller](#))
- Implement `dictGet` and `dictGetOrDefault` functions for Decimal types. [#5394](#) ([Artem Zuikov](#))

#### Improvement

- Debian init: Add service stop timeout [#5522](#) ([proller](#))
- Add setting `forbidden` by default to create table with suspicious types for LowCardinality [#5448](#) ([Olga Khvostikova](#))
- Regression functions return model weights when not used as State in function `evalMLMethod`. [#5411](#) ([Quid37](#))
- Rename and improve regression methods. [#5492](#) ([Quid37](#))
- Clearer interfaces of string searchers. [#5586](#) ([Danila Kutenin](#))

#### Bug Fix

- Fix potential data loss in Kafka [#5445](#) ([Ivan](#))
- Fix potential infinite loop in `PrettySpace` format when called with zero columns [#5560](#) ([Olga Khvostikova](#))

- Fixed UInt32 overflow bug in linear models. Allow eval ML model for non-const model argument. [#5516](#) ([Nikolai Kochetov](#))
- `ALTER TABLE ... DROP INDEX IF EXISTS ...` should not raise an exception if provided index does not exist [#5524](#) ([Gleb Novikov](#))
- Fix segfault with `bitmapHasAny` in scalar subquery [#5528](#) ([Zhichang Yu](#))
- Fixed error when replication connection pool does not retry to resolve host, even when DNS cache was dropped. [#5534](#) ([alesapin](#))
- Fixed `ALTER ... MODIFY TTL` on ReplicatedMergeTree. [#5539](#) ([Anton Popov](#))
- Fix INSERT into Distributed table with MATERIALIZED column [#5429](#) ([Azat Khuzhin](#))
- Fix bad alloc when truncate Join storage [#5437](#) ([TCeason](#))
- In recent versions of package tzdata some of files are symlinks now. The current mechanism for detecting default timezone gets broken and gives wrong names for some timezones. Now at least we force the timezone name to the contents of TZ if provided. [#5443](#) ([Ivan](#))
- Fix some extremely rare cases with MultiVolnitsky searcher when the constant needles in sum are at least 16KB long. The algorithm missed or overwrote the previous results which can lead to the incorrect result of `multiSearchAny`. [#5588](#) ([Danila Kutenin](#))
- Fix the issue when settings for ExternalData requests couldn't use ClickHouse settings. Also, for now, settings `date_time_input_format` and `low_cardinality_allow_in_native_format` cannot be used because of the ambiguity of names (in external data it can be interpreted as table format and in the query it can be a setting). [#5455](#) ([Danila Kutenin](#))
- Fix bug when parts were removed only from FS without dropping them from Zookeeper. [#5520](#) ([alesapin](#))
- Remove debug logging from MySQL protocol [#5478](#) ([alexey-milovidov](#))
- Skip ZNONODE during DDL query processing [#5489](#) ([Azat Khuzhin](#))
- Fix mix `UNION ALL` result column type. There were cases with inconsistent data and column types of resulting columns. [#5503](#) ([Artem Zuikov](#))
- Throw an exception on wrong integers in `dictGetT` functions instead of crash. [#5446](#) ([Artem Zuikov](#))
- Fix wrong `element_count` and `load_factor` for hashed dictionary in `system.dictionaries` table. [#5440](#) ([Azat Khuzhin](#))

## Build/Testing/Packaging Improvement

- Fixed build without Brotli HTTP compression support (`ENABLE_BROTLI=OFF` cmake variable). [#5521](#) ([Anton Yuzhaninov](#))
- Include roaring.h as roaring/roaring.h [#5523](#) ([Orivej Desh](#))
- Fix gcc9 warnings in hyperscan (#line directive is evil!) [#5546](#) ([Danila Kutenin](#))
- Fix all warnings when compiling with gcc-9. Fix some contrib issues. Fix gcc9 ICE and submit it to bugzilla. [#5498](#) ([Danila Kutenin](#))
- Fixed linking with lld [#5477](#) ([alexey-milovidov](#))
- Remove unused specializations in dictionaries [#5452](#) ([Artem Zuikov](#))

- Improvement performance tests for formatting and parsing tables for different types of files [#5497](#) ([Olga Khvostikova](#))
- Fixes for parallel test run [#5506](#) ([proller](#))
- Docker: use configs from clickhouse-test [#5531](#) ([proller](#))
- Fix compile for FreeBSD [#5447](#) ([proller](#))
- Upgrade boost to 1.70 [#5570](#) ([proller](#))
- Fix build clickhouse as submodule [#5574](#) ([proller](#))
- Improve JSONExtract performance tests [#5444](#) ([Vitaly Baranov](#))

## ClickHouse Release 19.8

### ClickHouse Release 19.8.3.8, 2019-06-11

#### New Features

- Added functions to work with JSON [#4686](#) ([hczi](#)) [#5124](#). ([Vitaly Baranov](#))
- Add a function basename, with a similar behaviour to a basename function, which exists in a lot of languages (`os.path.basename` in python, `basename` in PHP, etc...). Work with both an UNIX-like path or a Windows path. [#5136](#) ([Guillaume Tassery](#))
- Added `LIMIT n, m BY` or `LIMIT m OFFSET n BY` syntax to set offset of n for LIMIT BY clause. [#5138](#) ([Anton Popov](#))
- Added new data type `SimpleAggregateFunction`, which allows to have columns with light aggregation in an `AggregatingMergeTree`. This can only be used with simple functions like `any`, `anyLast`, `sum`, `min`, `max`. [#4629](#) ([Boris Granveaud](#))
- Added support for non-constant arguments in function `ngramDistance` [#5198](#) ([Danila Kutenin](#))
- Added functions `skewPop`, `skewSamp`, `kurtPop` and `kurtSamp` to compute for sequence skewness, sample skewness, kurtosis and sample kurtosis respectively. [#5200](#) ([hczi](#))
- Support rename operation for `MaterializeView` storage. [#5209](#) ([Guillaume Tassery](#))
- Added server which allows connecting to ClickHouse using MySQL client. [#4715](#) ([Yuriy Baranov](#))
- Add `toDecimal*OrZero` and `toDecimal*OrNull` functions. [#5291](#) ([Artem Zuikov](#))
- Support Decimal types in functions: `quantile`, `quantiles`, `median`, `quantileExactWeighted`, `quantilesExactWeighted`, `medianExactWeighted`. [#5304](#) ([Artem Zuikov](#))
- Added `toValidUTF8` function, which replaces all invalid UTF-8 characters by replacement character ♦ (U+FFFD). [#5322](#) ([Danila Kutenin](#))
- Added `format` function. Formatting constant pattern (simplified Python format pattern) with the strings listed in the arguments. [#5330](#) ([Danila Kutenin](#))
- Added `system.detached_parts` table containing information about detached parts of `MergeTree` tables. [#5353](#) ([akuzm](#))
- Added `ngramSearch` function to calculate the non-symmetric difference between needle and haystack. [#5418](#)[#5422](#) ([Danila Kutenin](#))

- Implementation of basic machine learning methods (stochastic linear regression and logistic regression) using aggregate functions interface. Has different strategies for updating model weights (simple gradient descent, momentum method, Nesterov method). Also supports mini-batches of custom size. [#4943 \(Quid37\)](#)
- Implementation of `geohashEncode` and `geohashDecode` functions. [#5003 \(Vasily Nemkov\)](#)
- Added aggregate function `timeSeriesGroupSum`, which can aggregate different time series that sample timestamp not alignment. It will use linear interpolation between two sample timestamp and then sum time-series together. Added aggregate function `timeSeriesGroupRateSum`, which calculates the rate of time-series and then sum rates together. [#4542 \(Yangkuan Liu\)](#)
- Added functions `IPv4CIDRtoIPv4Range` and `IPv6CIDRtoIPv6Range` to calculate the lower and higher bounds for an IP in the subnet using a CIDR. [#5095 \(Guillaume Tassery\)](#)
- Add a X-ClickHouse-Summary header when we send a query using HTTP with enabled setting `send_progress_in_http_headers`. Return the usual information of X-ClickHouse-Progress, with additional information like how many rows and bytes were inserted in the query. [#5116 \(Guillaume Tassery\)](#)

## Improvements

- Added `max_parts_in_total` setting for MergeTree family of tables (default: 100 000) that prevents unsafe specification of partition key [#5166. #5171 \(alexey-milovidov\)](#)
- `clickhouse-obfuscator`: derive seed for individual columns by combining initial seed with column name, not column position. This is intended to transform datasets with multiple related tables, so that tables will remain JOINable after transformation. [#5178 \(alexey-milovidov\)](#)
- Added functions `JSONExtractRaw`, `JSONExtractKeyAndValues`. Renamed functions `jsonExtract<type>` to `JSONExtract<type>`. When something goes wrong these functions return the correspondent values, not `NULL`. Modified function `JSONExtract`, now it gets the return type from its last parameter and does not inject nullables. Implemented fallback to RapidJSON in case AVX2 instructions are not available. Simdjson library updated to a new version. [#5235 \(Vitaly Baranov\)](#)
- Now `if` and `multilf` functions do not rely on the condition's `Nullable`, but rely on the branches for sql compatibility. [#5238 \(Jian Wu\)](#)
- `In` predicate now generates `Null` result from `Null` input like the `Equal` function. [#5152 \(Jian Wu\)](#)
- Check the time limit every (`flush_interval` / `poll_timeout`) number of rows from Kafka. This allows to break the reading from Kafka consumer more frequently and to check the time limits for the top-level streams [#5249 \(Ivan\)](#)
- Link rdkafka with bundled SASL. It should allow to use SASL SCRAM authentication [#5253 \(Ivan\)](#)
- Batched version of RowRefList for ALL JOINS. [#5267 \(Artem Zuikov\)](#)
- `clickhouse-server`: more informative listen error messages. [#5268 \(proller\)](#)
- Support dictionaries in `clickhouse-copier` for functions in `<sharding_key>` [#5270 \(proller\)](#)
- Add new setting `kafka_commit_every_batch` to regulate Kafka committing policy. It allows to set commit mode: after every batch of messages is handled, or after the whole block is written to the storage. It's a trade-off between losing some messages or reading them twice in some extreme situations. [#5308 \(Ivan\)](#)
- Make `windowFunnel` support other Unsigned Integer Types. [#5320 \(sundyli\)](#)
- Allow to shadow virtual column `_table` in Merge engine. [#5325 \(Ivan\)](#)
- Make `sequenceMatch` aggregate functions support other unsigned Integer types [#5339 \(sundyli\)](#)

- Better error messages if checksum mismatch is most likely caused by hardware failures. #5355 (alexey-milovidov)
- Check that underlying tables support sampling for StorageMerge #5366 (Ivan)
- Close MySQL connections after their usage in external dictionaries. It is related to issue #893. #5395 (Clément Rodriguez)
- Improvements of MySQL Wire Protocol. Changed name of format to MySQLWire. Using RAI<sup>I</sup> for calling RSA\_free. Disabling SSL if context cannot be created. #5419 (Yuriy Baranov)
- clickhouse-client: allow to run with unaccessible history file (read-only, no disk space, file is directory, ...). #5431 (proller)
- Respect query settings in asynchronous INSERTs into Distributed tables. #4936 (TCeason)
- Renamed functions leastSqr to simpleLinearRegression, LinearRegression to linearRegression, LogisticRegression to logisticRegression. #5391 (Nikolai Kochetov)

## Performance Improvements

- Parallelize processing of parts of non-replicated MergeTree tables in ALTER MODIFY query. #4639 (Ivan Kush)
- Optimizations in regular expressions extraction. #5193 #5191 (Danila Kutenin)
- Do not add right join key column to join result if it's used only in join on section. #5260 (Artem Zuikov)
- Freeze the Kafka buffer after first empty response. It avoids multiple invocations of ReadBuffer::next() for empty result in some row-parsing streams. #5283 (Ivan)
- concat function optimization for multiple arguments. #5357 (Danila Kutenin)
- Query optimisation. Allow push down IN statement while rewriting comma/cross join into inner one. #5396 (Artem Zuikov)
- Upgrade our LZ4 implementation with reference one to have faster decompression. #5070 (Danila Kutenin)
- Implemented MSD radix sort (based on kxsort), and partial sorting. #5129 (Evgenii Pravda)

## Bug Fixes

- Fix push require columns with join #5192 (Winter Zhang)
- Fixed bug, when ClickHouse is run by systemd, the command sudo service clickhouse-server forcerestart was not working as expected. #5204 (proller)
- Fix http error codes in DataPartsExchange (interserver http server on 9009 port always returned code 200, even on errors). #5216 (proller)
- Fix SimpleAggregateFunction for String longer than MAX\_SMALL\_STRING\_SIZE #5311 (Azat Khuzhin)
- Fix error for Decimal to Nullable(Decimal) conversion in IN. Support other Decimal to Decimal conversions (including different scales). #5350 (Artem Zuikov)
- Fixed FPU clobbering in simdjson library that lead to wrong calculation of uniqHLL and uniqCombined aggregate function and math functions such as log. #5354 (alexey-milovidov)
- Fixed handling mixed const/nonconst cases in JSON functions. #5435 (Vitaly Baranov)

- Fix retention function. Now all conditions that satisfy in a row of data are added to the data state. #5119 (小路)
- Fix result type for quantileExact with Decimals. #5304 (Artem Zuikov)

## Documentation

- Translate documentation for CollapsingMergeTree to chinese. #5168 (张风啸)
- Translate some documentation about table engines to chinese.  
#5134  
#5328  
(never lee)

## Build/Testing/Packaging Improvements

- Fix some sanitizer reports that show probable use-after-free. #5139 #5143 #5393 (Ivan)
- Move performance tests out of separate directories for convenience. #5158 (alexey-milovidov)
- Fix incorrect performance tests. #5255 (alesapin)
- Added a tool to calculate checksums caused by bit flips to debug hardware issues. #5334 (alexey-milovidov)
- Make runner script more usable. #5340#5360 (filimonov)
- Add small instruction how to write performance tests. #5408 (alesapin)
- Add ability to make substitutions in create, fill and drop query in performance tests #5367 (Olga Khvostikova)

# ClickHouse Release 19.7

## ClickHouse Release 19.7.5.29, 2019-07-05

### Bug Fix

- Fix performance regression in some queries with JOIN. #5192 (Winter Zhang)

## ClickHouse Release 19.7.5.27, 2019-06-09

### New Features

- Added bitmap related functions bitmapHasAny and bitmapHasAll analogous to hasAny and hasAll functions for arrays. #5279 (Sergi Vladynkin)

### Bug Fixes

- Fix segfault on minmax INDEX with Null value. #5246 (Nikita Vasilev)
- Mark all input columns in LIMIT BY as required output. It fixes ‘Not found column’ error in some distributed queries. #5407 (Constantin S. Pan)
- Fix “Column ‘0’ already exists” error in SELECT .. PREWHERE on column with DEFAULT #5397 (proller)
- Fix ALTER MODIFY TTL query on ReplicatedMergeTree. #5539 (Anton Popov)
- Don’t crash the server when Kafka consumers have failed to start. #5285 (Ivan)
- Fixed bitmap functions produce wrong result. #5359 (Andy Yang)
- Fix element\_count for hashed dictionary (do not include duplicates) #5440 (Azat Khuzhin)

- Use contents of environment variable TZ as the name for timezone. It helps to correctly detect default timezone in some cases. [#5443](#) ([Ivan](#))
- Do not try to convert integers in `dictGetT` functions, because it does not work correctly. Throw an exception instead. [#5446](#) ([Artem Zuikov](#))
- Fix settings in ExternalData HTTP request. [#5455](#) ([Danila Kutenin](#))
- Fix bug when parts were removed only from FS without dropping them from Zookeeper. [#5520](#) ([alesapin](#))
- Fix segmentation fault in `bitmapHasAny` function. [#5528](#) ([Zhichang Yu](#))
- Fixed error when replication connection pool does not retry to resolve host, even when DNS cache was dropped. [#5534](#) ([alesapin](#))
- Fixed `DROP INDEX IF EXISTS` query. Now `ALTER TABLE ... DROP INDEX IF EXISTS ...` query does not raise an exception if provided index does not exist. [#5524](#) ([Gleb Novikov](#))
- Fix union all supertype column. There were cases with inconsistent data and column types of resulting columns. [#5503](#) ([Artem Zuikov](#))
- Skip ZNONODE during DDL query processing. Before if another node removes the znode in task queue, the one that did not process it, but already get list of children, will terminate the DDLWorker thread. [#5489](#) ([Azat Khuzhin](#))
- Fix `INSERT` into `Distributed()` table with `MATERIALIZED` column. [#5429](#) ([Azat Khuzhin](#))

## ClickHouse Release 19.7.3.9, 2019-05-30

### New Features

- Allow to limit the range of a setting that can be specified by user. These constraints can be set up in user settings profile. [#4931](#) ([Vitaly Baranov](#))
- Add a second version of the function `groupUniqArray` with an optional `max_size` parameter that limits the size of the resulting array. This behavior is similar to `groupArray(max_size)(x)` function. [#5026](#) ([Guillaume Tassery](#))
- For `TSVWithNames`/`CSVWithNames` input file formats, column order can now be determined from file header. This is controlled by `input_format_with_names_use_header` parameter. [#5081](#) ([Alexander](#))

### Bug Fixes

- Crash with `uncompressed_cache + JOIN` during merge ([#5197](#))  
[#5133](#) ([Danila Kutenin](#))
- Segmentation fault on a `clickhouse-client` query to system tables. [#5066](#)  
[#5127](#) ([Ivan](#))

- Data loss on heavy load via KafkaEngine (#4736)  
[#5080](#)  
(Ivan)
- Fixed very rare data race condition that could happen when executing a query with UNION ALL involving at least two SELECTs from system.columns, system.tables, system.parts, system.parts\_tables or tables of Merge family and performing ALTER of columns of the related tables concurrently. [#5189](#) (alexey-milovidov)

## Performance Improvements

- Use radix sort for sorting by single numeric column in ORDER BY without LIMIT. [#5106](#),  
[#4439](#)  
(Evgenii Pravda,  
alexey-milovidov)

## Documentation

- Translate documentation for some table engines to Chinese.  
[#5107](#),  
[#5094](#),  
[#5087](#)  
(张风啸),  
[#5068](#) (never  
lee)

## Build/Testing/Packaging Improvements

- Print UTF-8 characters properly in clickhouse-test.  
[#5084](#)  
(alexey-milovidov)
- Add command line parameter for clickhouse-client to always load suggestion data. [#5102](#)  
(alexey-milovidov)
- Resolve some of PVS-Studio warnings.  
[#5082](#)  
(alexey-milovidov)
- Update LZ4 [#5040](#) (Danila Kutenin)
- Add gperf to build requirements for upcoming pull request #5030.  
[#5110](#)  
(proller)

# ClickHouse Release 19.6

## ClickHouse Release 19.6.3.18, 2019-06-13

### Bug Fixes

- Fixed IN condition pushdown for queries from table functions mysql and odbc and corresponding table engines. This fixes #3540 and #2384. [#5313](#) (alexey-milovidov)
- Fix deadlock in Zookeeper. [#5297](#) (github1youlc)
- Allow quoted decimals in CSV. [#5284](#) (Artem Zuikov)

- Disallow conversion from float Inf/NaN into Decimals (throw exception). [#5282](#) ([Artem Zuikov](#))
- Fix data race in rename query. [#5247](#) ([Winter Zhang](#))
- Temporarily disable LFAlloc. Usage of LFAlloc might lead to a lot of MAP\_FAILED in allocating UncompressedCache and in a result to crashes of queries at high loaded servers. [cfdba93](#)([Danila Kutenin](#))

## ClickHouse Release 19.6.2.11, 2019-05-13

### New Features

- TTL expressions for columns and tables. [#4212](#) ([Anton Popov](#))
- Added support for `brotli` compression for HTTP responses (Accept-Encoding: br) [#4388](#) ([Mikhail](#))
- Added new function `isValidUTF8` for checking whether a set of bytes is correctly utf-8 encoded. [#4934](#) ([Danila Kutenin](#))
- Add new load balancing policy `first_or_random` which sends queries to the first specified host and if it's inaccessible send queries to random hosts of shard. Useful for cross-replication topology setups. [#5012](#) ([nvartolomei](#))

### Experimental Features

- Add setting `index_granularity_bytes` (adaptive index granularity) for MergeTree\* tables family. [#4826](#) ([alesapin](#))

### Improvements

- Added support for non-constant and negative size and length arguments for function `substringUTF8`. [#4989](#) ([alexey-milovidov](#))
- Disable push-down to right table in left join, left table in right join, and both tables in full join. This fixes wrong JOIN results in some cases. [#4846](#) ([Ivan](#))
- `clickhouse-copier`: auto upload task configuration from `--task-file` option [#4876](#) ([proller](#))
- Added typos handler for storage factory and table functions factory. [#4891](#) ([Danila Kutenin](#))
- Support asterisks and qualified asterisks for multiple joins without subqueries [#4898](#) ([Artem Zuikov](#))
- Make missing column error message more user friendly. [#4915](#) ([Artem Zuikov](#))

### Performance Improvements

- Significant speedup of ASOF JOIN [#4924](#) ([Martijn Bakker](#))

### Backward Incompatible Changes

- HTTP header `Query-Id` was renamed to `X-ClickHouse-Query-Id` for consistency. [#4972](#) ([Mikhail](#))

### Bug Fixes

- Fixed potential null pointer dereference in `clickhouse-copier`. [#4900](#) ([proller](#))
- Fixed error on query with JOIN + ARRAY JOIN [#4938](#) ([Artem Zuikov](#))
- Fixed hanging on start of the server when a dictionary depends on another dictionary via a database with engine=Dictionary. [#4962](#) ([Vitaly Baranov](#))
- 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](#))

- Fix potentially wrong result for `SELECT DISTINCT` with `JOIN` #5001 (Artem Zuikov)
- Fixed very rare data race condition that could happen when executing a query with `UNION ALL` involving at least two `SELECTs` from `system.columns`, `system.tables`, `system.parts`, `system.parts_tables` or tables of Merge family and performing `ALTER` of columns of the related tables concurrently. #5189 (alexey-milovidov)

## Build/Testing/Packaging Improvements

- Fixed test failures when running `clickhouse-server` on different host #4713 (Vasily Nemkov)
- `clickhouse-test`: Disable color control sequences in non tty environment. #4937 (alesapin)
- `clickhouse-test`: Allow use any test database (remove `test.` qualification where it possible) #5008 (proller)
- Fix ubsan errors #5037 (Vitaly Baranov)
- Yandex LFAlloc was added to ClickHouse to allocate `MarkCache` and `UncompressedCache` data in different ways to catch segfaults more reliable #4995 (Danila Kutenin)
- Python util to help with backports and changelogs. #4949 (Ivan)

# ClickHouse Release 19.5

## ClickHouse Release 19.5.4.22, 2019-05-13

### Bug Fixes

- Fixed possible crash in `bitmap*` functions #5220 #5228 (Andy Yang)
- Fixed very rare data race condition that could happen when executing a query with `UNION ALL` involving at least two `SELECTs` from `system.columns`, `system.tables`, `system.parts`, `system.parts_tables` or tables of Merge family and performing `ALTER` of columns of the related tables concurrently. #5189 (alexey-milovidov)
- Fixed error `Set for IN is not created yet` in case of using single `LowCardinality` column in the left part of `IN`. This error happened if `LowCardinality` column was the part of primary key. #5031 #5154 (Nikolai Kochetov)
- Modification of retention function: If a row satisfies both the first and NTH condition, only the first satisfied condition is added to the data state. Now all conditions that satisfy in a row of data are added to the data state. #5119 (小路)

## ClickHouse Release 19.5.3.8, 2019-04-18

### Bug Fixes

- Fixed type of setting `max_partitions_per_insert_block` from boolean to `UInt64`. #5028 (Mohammad Hossein Sekhavat)

## ClickHouse Release 19.5.2.6, 2019-04-15

### New Features

- `Hyperscan` multiple regular expression matching was added (functions `multiMatchAny`, `multiMatchAnyIndex`, `multiFuzzyMatchAny`, `multiFuzzyMatchAnyIndex`). #4780, #4841 (Danila Kutenin)
- `multiSearchFirstPosition` function was added. #4780 (Danila Kutenin)
- Implement the predefined expression filter per row for tables. #4792 (Ivan)
- A new type of data skipping indices based on bloom filters (can be used for `equal`, `in` and `like` functions). #4499 (Nikita Vasilev)

- Added ASOF JOIN which allows to run queries that join to the most recent value known. #4774 #4867 #4863 #4875 (Martijn Bakker, Artem Zuikov)
- Rewrite multiple COMMA JOIN to CROSS JOIN. Then rewrite them to INNER JOIN if possible. #4661 (Artem Zuikov)

## Improvement

- `topK` and `topKWeighted` now supports custom `loadFactor` (fixes issue #4252). #4634 (Kirill Danshin)
- Allow to use `parallel_replicas_count > 1` even for tables without sampling (the setting is simply ignored for them). In previous versions it was lead to exception. #4637 (Alexey Elymanov)
- Support for `CREATE OR REPLACE VIEW`. Allow to create a view or set a new definition in a single statement. #4654 (Boris Granveaud)
- Buffer table engine now supports `PREWHERE`. #4671 (Yangkuan Liu)
- Add ability to start replicated table without metadata in zookeeper in `readonly` mode. #4691 (alesapin)
- Fixed flicker of progress bar in clickhouse-client. The issue was most noticeable when using `FORMAT Null` with streaming queries. #4811 (alexey-milovidov)
- Allow to disable functions with `hyperscan` library on per user basis to limit potentially excessive and uncontrolled resource usage. #4816 (alexey-milovidov)
- Add version number logging in all errors. #4824 (proller)
- Added restriction to the `multiMatch` functions which requires string size to fit into `unsigned int`. Also added the number of arguments limit to the `multiSearch` functions. #4834 (Danila Kutenin)
- Improved usage of scratch space and error handling in Hyperscan. #4866 (Danila Kutenin)
- Fill `system.graphite_detentions` from a table config of \*GraphiteMergeTree engine tables. #4584 (Mikhail f. Shiryaev)
- Rename `trigramDistance` function to `ngramDistance` and add more functions with `CaseInsensitive` and `UTF`. #4602 (Danila Kutenin)
- Improved data skipping indices calculation. #4640 (Nikita Vasilev)
- Keep ordinary, DEFAULT, MATERIALIZED and ALIAS columns in a single list (fixes issue #2867). #4707 (Alex Zatelepin)

## Bug Fix

- Avoid `std::terminate` in case of memory allocation failure. Now `std::bad_alloc` exception is thrown as expected. #4665 (alexey-milovidov)
- Fixes capnproto reading from buffer. Sometimes files wasn't loaded successfully by HTTP. #4674 (Vladislav)
- Fix error Unknown log entry type: 0 after `OPTIMIZE TABLE FINAL` query. #4683 (Amos Bird)
- Wrong arguments to `hasAny` or `hasAll` functions may lead to segfault. #4698 (alexey-milovidov)
- Deadlock may happen while executing `DROP DATABASE` dictionary query. #4701 (alexey-milovidov)
- Fix undefined behavior in `median` and `quantile` functions. #4702 (hcjz)
- Fix compression level detection when `network_compression_method` in lowercase. Broken in v19.1. #4706 (proller)

- Fixed ignorance of <timezone>UTC</timezone> setting (fixes issue #4658). #4718 (proller)
- Fix histogram function behaviour with Distributed tables. #4741 (olegkv)
- Fixed tsan report destroy of a locked mutex. #4742 (alexey-milovidov)
- Fixed TSan report on shutdown due to race condition in system logs usage. Fixed potential use-after-free on shutdown when part\_log is enabled. #4758 (alexey-milovidov)
- Fix recheck parts in ReplicatedMergeTreeAlterThread in case of error. #4772 (Nikolai Kochetov)
- Arithmetic operations on intermediate aggregate function states were not working for constant arguments (such as subquery results). #4776 (alexey-milovidov)
- Always backquote column names in metadata. Otherwise it's impossible to create a table with column named index (server won't restart due to malformed ATTACH query in metadata). #4782 (alexey-milovidov)
- Fix crash in ALTER ... MODIFY ORDER BY on Distributed table. #4790 (TCeason)
- Fix segfault in JOIN ON with enabled enable\_optimize\_predicate\_expression. #4794 (Winter Zhang)
- Fix bug with adding an extraneous row after consuming a protobuf message from Kafka. #4808 (Vitaly Baranov)
- Fix crash of JOIN on not-nullable vs nullable column. Fix NULLs in right keys in ANY JOIN + join\_use\_nulls. #4815 (Artem Zuikov)
- Fix segmentation fault in clickhouse-copier. #4835 (proller)
- Fixed race condition in SELECT from system.tables if the table is renamed or altered concurrently. #4836 (alexey-milovidov)
- Fixed data race when fetching data part that is already obsolete. #4839 (alexey-milovidov)
- Fixed rare data race that can happen during RENAME table of MergeTree family. #4844 (alexey-milovidov)
- Fixed segmentation fault in function arrayIntersect. Segmentation fault could happen if function was called with mixed constant and ordinary arguments. #4847 (Lixiang Qian)
- Fixed reading from Array(LowCardinality) column in rare case when column contained a long sequence of empty arrays. #4850 (Nikolai Kochetov)
- Fix crash in FULL/RIGHT JOIN when we joining on nullable vs not nullable. #4855 (Artem Zuikov)
- Fix No message received exception while fetching parts between replicas. #4856 (alesapin)
- Fixed arrayIntersect function wrong result in case of several repeated values in single array. #4871 (Nikolai Kochetov)
- Fix a race condition during concurrent ALTER COLUMN queries that could lead to a server crash (fixes issue #3421). #4592 (Alex Zatelepin)
- Fix incorrect result in FULL/RIGHT JOIN with const column. #4723 (Artem Zuikov)
- Fix duplicates in GLOBAL JOIN with asterisk. #4705 (Artem Zuikov)
- Fix parameter deduction in ALTER MODIFY of column CODEC when column type is not specified. #4883 (alesapin)

- Functions `cutQueryStringAndFragment()` and `queryStringAndFragment()` now works correctly when `URL` contains a fragment and no query. [#4894 \(Vitaly Baranov\)](#)
- Fix rare bug when setting `min_bytes_to_use_direct_io` is greater than zero, which occurs when thread have to seek backward in column file. [#4897 \(alesapin\)](#)
- Fix wrong argument types for aggregate functions with `LowCardinality` arguments (fixes issue [#4919](#)). [#4922 \(Nikolai Kochetov\)](#)
- Fix wrong name qualification in `GLOBAL JOIN`. [#4969 \(Artem Zuikov\)](#)
- Fix function `toISOWeek` result for year 1970. [#4988 \(alexey-milovidov\)](#)
- Fix `DROP`, `TRUNCATE` and `OPTIMIZE` queries duplication, when executed on `ON CLUSTER` for `ReplicatedMergeTree*` tables family. [#4991 \(alesapin\)](#)

## Backward Incompatible Change

- Rename setting `insert_sample_with_metadata` to setting `input_format_defaults_for_omitted_fields`. [#4771 \(Artem Zuikov\)](#)
- Added setting `max_partitions_per_insert_block` (with value 100 by default). If inserted block contains larger number of partitions, an exception is thrown. Set it to 0 if you want to remove the limit (not recommended). [#4845 \(alexey-milovidov\)](#)
- Multi-search functions were renamed (`multiPosition` to `multiSearchAllPositions`, `multiSearch` to `multiSearchAny`, `firstMatch` to `multiSearchFirstIndex`). [#4780 \(Danila Kutenin\)](#)

## Performance Improvement

- Optimize Volnitsky searcher by inlining, giving about 5-10% search improvement for queries with many needles or many similar bigrams. [#4862 \(Danila Kutenin\)](#)
- Fix performance issue when setting `use_uncompressed_cache` is greater than zero, which appeared when all read data contained in cache. [#4913 \(alesapin\)](#)

## Build/Testing/Packaging Improvement

- Hardening debug build: more granular memory mappings and ASLR; add memory protection for mark cache and index. This allows to find more memory stomping bugs in case when ASan and MSan cannot do it. [#4632 \(alexey-milovidov\)](#)
- Add support for cmake variables `ENABLE_PROTOBUF`, `ENABLE_PARQUET` and `ENABLE_BROTLI` which allows to enable/disable the above features (same as we can do for librdkafka, mysql, etc). [#4669 \(Silviu Caragea\)](#)
- Add ability to print process list and stacktraces of all threads if some queries are hung after test run. [#4675 \(alesapin\)](#)
- Add retries on `Connection loss` error in `clickhouse-test`. [#4682 \(alesapin\)](#)
- Add freebsd build with vagrant and build with thread sanitizer to packager script. [#4712 #4748 \(alesapin\)](#)
- Now user asked for password for user '`default`' during installation. [#4725 \(proller\)](#)
- Suppress warning in `rdkafka` library. [#4740 \(alexey-milovidov\)](#)
- Allow ability to build without ssl. [#4750 \(proller\)](#)
- Add a way to launch `clickhouse-server` image from a custom user. [#4753 \(Mikhail f. Shiryaev\)](#)

- Upgrade contrib boost to 1.69. [#4793](#) ([proller](#))
- Disable usage of `mremap` when compiled with Thread Sanitizer. Surprisingly enough, TSan does not intercept `mremap` (though it does intercept `mmap`, `munmap`) that leads to false positives. Fixed TSan report in stateful tests. [#4859](#) ([alexey-milovidov](#))
- Add test checking using format schema via HTTP interface. [#4864](#) ([Vitaly Baranov](#))

## ClickHouse Release 19.4

### ClickHouse Release 19.4.4.33, 2019-04-17

#### Bug Fixes

- Avoid `std::terminate` in case of memory allocation failure. Now `std::bad_alloc` exception is thrown as expected. [#4665](#) ([alexey-milovidov](#))
- Fixes capnproto reading from buffer. Sometimes files wasn't loaded successfully by HTTP. [#4674](#) ([Vladislav](#))
- Fix error Unknown log entry type: 0 after `OPTIMIZE TABLE FINAL` query. [#4683](#) ([Amos Bird](#))
- Wrong arguments to `hasAny` or `hasAll` functions may lead to segfault. [#4698](#) ([alexey-milovidov](#))
- Deadlock may happen while executing `DROP DATABASE` dictionary query. [#4701](#) ([alexey-milovidov](#))
- Fix undefined behavior in `median` and `quantile` functions. [#4702](#) ([hcz](#))
- Fix compression level detection when `network_compression_method` in lowercase. Broken in v19.1. [#4706](#) ([proller](#))
- Fixed ignorance of `<timezone>UTC</timezone>` setting (fixes issue [#4658](#)). [#4718](#) ([proller](#))
- Fix `histogram` function behaviour with `Distributed` tables. [#4741](#) ([olegkv](#))
- Fixed tsan report `destroy of a locked mutex`. [#4742](#) ([alexey-milovidov](#))
- Fixed TSan report on shutdown due to race condition in system logs usage. Fixed potential use-after-free on shutdown when `part_log` is enabled. [#4758](#) ([alexey-milovidov](#))
- Fix recheck parts in `ReplicatedMergeTreeAlterThread` in case of error. [#4772](#) ([Nikolai Kochetov](#))
- Arithmetic operations on intermediate aggregate function states were not working for constant arguments (such as subquery results). [#4776](#) ([alexey-milovidov](#))
- Always backquote column names in metadata. Otherwise it's impossible to create a table with column named `index` (server won't restart due to malformed `ATTACH` query in metadata). [#4782](#) ([alexey-milovidov](#))
- Fix crash in `ALTER ... MODIFY ORDER BY` on `Distributed` table. [#4790](#) ([TCeason](#))
- Fix segfault in `JOIN ON` with enabled `enable_optimize_predicate_expression`. [#4794](#) ([Winter Zhang](#))
- Fix bug with adding an extraneous row after consuming a protobuf message from Kafka. [#4808](#) ([Vitaly Baranov](#))
- Fix segmentation fault in `clickhouse-copier`. [#4835](#) ([proller](#))
- Fixed race condition in `SELECT` from `system.tables` if the table is renamed or altered concurrently. [#4836](#) ([alexey-milovidov](#))
- Fixed data race when fetching data part that is already obsolete. [#4839](#) ([alexey-milovidov](#))

- Fixed rare data race that can happen during `RENAME` table of MergeTree family. #4844 (alexey-milovidov)
- Fixed segmentation fault in function `arrayIntersect`. Segmentation fault could happen if function was called with mixed constant and ordinary arguments. #4847 (Lixiang Qian)
- Fixed reading from `Array(LowCardinality)` column in rare case when column contained a long sequence of empty arrays. #4850 (Nikolai Kochetov)
- Fix `No message received` exception while fetching parts between replicas. #4856 (alesapin)
- Fixed `arrayIntersect` function wrong result in case of several repeated values in single array. #4871 (Nikolai Kochetov)
- Fix a race condition during concurrent `ALTER COLUMN` queries that could lead to a server crash (fixes issue #3421). #4592 (Alex Zatelepin)
- Fix parameter deduction in `ALTER MODIFY` of column `CODEC` when column type is not specified. #4883 (alesapin)
- Functions `cutQueryStringAndFragment()` and `queryStringAndFragment()` now works correctly when `URL` contains a fragment and no query. #4894 (Vitaly Baranov)
- Fix rare bug when setting `min_bytes_to_use_direct_io` is greater than zero, which occurs when thread have to seek backward in column file. #4897 (alesapin)
- Fix wrong argument types for aggregate functions with `LowCardinality` arguments (fixes issue #4919). #4922 (Nikolai Kochetov)
- Fix function `toISOWeek` result for year 1970. #4988 (alexey-milovidov)
- Fix `DROP`, `TRUNCATE` and `OPTIMIZE` queries duplication, when executed on `ON CLUSTER` for `ReplicatedMergeTree*` tables family. #4991 (alesapin)

## Improvements

- Keep ordinary, `DEFAULT`, `MATERIALIZED` and `ALIAS` columns in a single list (fixes issue #2867). #4707 (Alex Zatelepin)

## ClickHouse Release 19.4.3.11, 2019-04-02

### Bug Fixes

- Fix crash in `FULL/RIGHT JOIN` when we joining on nullable vs not nullable. #4855 (Artem Zuikov)
- Fix segmentation fault in `clickhouse-copier`. #4835 (proller)

### Build/Testing/Packaging Improvement

- Add a way to launch `clickhouse-server` image from a custom user. #4753 (Mikhail f. Shiryaev)

## ClickHouse Release 19.4.2.7, 2019-03-30

### Bug Fixes

- Fixed reading from `Array(LowCardinality)` column in rare case when column contained a long sequence of empty arrays. #4850 (Nikolai Kochetov)

## ClickHouse Release 19.4.1.3, 2019-03-19

### Bug Fixes

- Fixed remote queries which contain both `LIMIT BY` and `LIMIT`. Previously, if `LIMIT BY` and `LIMIT` were used for remote query, `LIMIT` could happen before `LIMIT BY`, which led to too filtered result. [#4708](#) ([Constantin S. Pan](#))

## ClickHouse Release 19.4.0.49, 2019-03-09

### New Features

- Added full support for `Protobuf` format (input and output, nested data structures). [#4174](#) [#4493](#) ([Vitaly Baranov](#))
- Added bitmap functions with Roaring Bitmaps. [#4207](#) ([Andy Yang](#)) [#4568](#) ([Vitaly Baranov](#))
- Parquet format support. [#4448](#) ([proller](#))
- N-gram distance was added for fuzzy string comparison. It is similar to q-gram metrics in R language. [#4466](#) ([Danila Kutenin](#))
- Combine rules for graphite rollup from dedicated aggregation and retention patterns. [#4426](#) ([Mikhail f. Shiryaev](#))
- Added `max_execution_speed` and `max_execution_speed_bytes` to limit resource usage. Added `min_execution_speed_bytes` setting to complement the `min_execution_speed`. [#4430](#) ([Winter Zhang](#))
- Implemented function `flatten`. [#4555](#) [#4409](#) ([alexey-milovidov](#), [kzon](#))
- Added functions `arrayEnumerateDenseRanked` and `arrayEnumerateUniqRanked` (it's like `arrayEnumerateUniq` but allows to fine tune array depth to look inside multidimensional arrays). [#4475](#) ([proller](#)) [#4601](#) ([alexey-milovidov](#))
- Multiple JOINS with some restrictions: no asterisks, no complex aliases in ON/WHERE/GROUP BY/... [#4462](#) ([Artem Zuikov](#))

### Bug Fixes

- This release also contains all bug fixes from 19.3 and 19.1.
- Fixed bug in data skipping indices: order of granules after `INSERT` was incorrect. [#4407](#) ([Nikita Vasilev](#))
- Fixed `set` index for `Nullable` and `LowCardinality` columns. Before it, `set` index with `Nullable` or `LowCardinality` column led to error `Data type must be deserialized with multiple streams while selecting`. [#4594](#) ([Nikolai Kochetov](#))
- Correctly set `update_time` on full `executable` dictionary update. [#4551](#) ([Tema Novikov](#))
- Fix broken progress bar in 19.3. [#4627](#) ([filimonov](#))
- Fixed inconsistent values of `MemoryTracker` when memory region was shrunked, in certain cases. [#4619](#) ([alexey-milovidov](#))
- Fixed undefined behaviour in `ThreadPool`. [#4612](#) ([alexey-milovidov](#))
- Fixed a very rare crash with the message `mutex lock failed: Invalid argument` that could happen when a `MergeTree` table was dropped concurrently with a `SELECT`. [#4608](#) ([Alex Zatelepin](#))
- ODBC driver compatibility with `LowCardinality` data type. [#4381](#) ([proller](#))
- FreeBSD: Fixup for `AIOContextPool`: Found `io_event` with unknown id 0 error. [#4438](#) ([urgordeadbeef](#))
- `system.part_log` table was created regardless to configuration. [#4483](#) ([alexey-milovidov](#))
- Fix undefined behaviour in `dictIsIn` function for cache dictionaries. [#4515](#) ([alesapin](#))

- 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 \(Alex Zatelepin\)](#)
- Disable compile\_expressions by default until we get own llvm contrib and can test it with clang and asan. [#4579 \(alesapin\)](#)
- Prevent std::terminate when invalidate\_query for clickhouse external dictionary source has returned wrong resultset (empty or more than one row or more than one column). Fixed issue when the invalidate\_query was performed every five seconds regardless to the lifetime. [#4583 \(alexey-milovidov\)](#)
- Avoid deadlock when the invalidate\_query for a dictionary with clickhouse source was involving system.dictionaries table or Dictionaries database (rare case). [#4599 \(alexey-milovidov\)](#)
- Fixes for CROSS JOIN with empty WHERE. [#4598 \(Artem Zuikov\)](#)
- Fixed segfault in function “replicate” when constant argument is passed. [#4603 \(alexey-milovidov\)](#)
- Fix lambda function with predicate optimizer. [#4408 \(Winter Zhang\)](#)
- Multiple JOINs multiple fixes. [#4595 \(Artem Zuikov\)](#)

## Improvements

- Support aliases in JOIN ON section for right table columns. [#4412 \(Artem Zuikov\)](#)
- Result of multiple JOINs need correct result names to be used in subselects. Replace flat aliases with source names in result. [#4474 \(Artem Zuikov\)](#)
- Improve push-down logic for joined statements. [#4387 \(Ivan\)](#)

## Performance Improvements

- Improved heuristics of “move to PREWHERE” optimization. [#4405 \(alexey-milovidov\)](#)
- Use proper lookup tables that uses HashTable’s API for 8-bit and 16-bit keys. [#4536 \(Amos Bird\)](#)
- Improved performance of string comparison. [#4564 \(alexey-milovidov\)](#)
- Cleanup distributed DDL queue in a separate thread so that it does not slow down the main loop that processes distributed DDL tasks. [#4502 \(Alex Zatelepin\)](#)
- When min\_bytes\_to\_use\_direct\_io is set to 1, not every file was opened with O\_DIRECT mode because the data size to read was sometimes underestimated by the size of one compressed block. [#4526 \(alexey-milovidov\)](#)

## Build/Testing/Packaging Improvement

- Added support for clang-9 [#4604 \(alexey-milovidov\)](#)
- Fix wrong \_\_asm\_\_ instructions (again) [#4621 \(Konstantin Podshumok\)](#)
- Add ability to specify settings for clickhouse-performance-test from command line. [#4437 \(alesapin\)](#)
- Add dictionaries tests to integration tests. [#4477 \(alesapin\)](#)
- Added queries from the benchmark on the website to automated performance tests. [#4496 \(alexey-milovidov\)](#)
- xxhash.h does not exist in external lz4 because it is an implementation detail and its symbols are namespaced with XXH\_NAMESPACE macro. When lz4 is external, xxHash has to be external too, and the dependents have to link to it. [#4495 \(Orivej Desh\)](#)

- Fixed a case when `quantileTiming` aggregate function can be called with negative or floating point argument (this fixes fuzz test with undefined behaviour sanitizer). [#4506 \(alexey-milovidov\)](#)
- Spelling error correction. [#4531 \(sdk2\)](#)
- Fix compilation on Mac. [#4371 \(Vitaly Baranov\)](#)
- Build fixes for FreeBSD and various unusual build configurations. [#4444 \(proller\)](#)

## ClickHouse Release 19.3

### ClickHouse Release 19.3.9.1, 2019-04-02

#### Bug Fixes

- Fix crash in `FULL/RIGHT JOIN` when we joining on nullable vs not nullable. [#4855 \(Artem Zuikov\)](#)
- Fix segmentation fault in `clickhouse-copier`. [#4835 \(proller\)](#)
- Fixed reading from `Array(LowCardinality)` column in rare case when column contained a long sequence of empty arrays. [#4850 \(Nikolai Kochetov\)](#)

#### Build/Testing/Packaging Improvement

- Add a way to launch `clickhouse-server` image from a custom user [#4753 \(Mikhail f. Shiryaev\)](#)

### ClickHouse Release 19.3.7, 2019-03-12

#### Bug Fixes

- Fixed error in #3920. This error manifests itself as random cache corruption (messages `Unknown codec family code, Cannot seek through file`) and segfaults. This bug first appeared in version 19.1 and is present in versions up to 19.1.10 and 19.3.6. [#4623 \(alexey-milovidov\)](#)

### ClickHouse Release 19.3.6, 2019-03-02

#### Bug Fixes

- When there are more than 1000 threads in a thread pool, `std::terminate` may happen on thread exit. [Azat Khuzhin #4485 #4505 \(alexey-milovidov\)](#)
- Now it's possible to create `ReplicatedMergeTree*` tables with comments on columns without defaults and tables with columns codecs without comments and defaults. Also fix comparison of codecs. [#4523 \(alesapin\)](#)
- Fixed crash on JOIN with array or tuple. [#4552 \(Artem Zuikov\)](#)
- Fixed crash in `clickhouse-copier` with the message `ThreadStatus not created`. [#4540 \(Artem Zuikov\)](#)
- Fixed hangup on server shutdown if distributed DDLs were used. [#4472 \(Alex Zatelepin\)](#)
- Incorrect column numbers were printed in error message about text format parsing for columns with number greater than 10. [#4484 \(alexey-milovidov\)](#)

#### Build/Testing/Packaging Improvements

- Fixed build with AVX enabled. [#4527 \(alexey-milovidov\)](#)
- Enable extended accounting and IO accounting based on good known version instead of kernel under which it is compiled. [#4541 \(nvartolomei\)](#)
- Allow to skip setting of `core_dump.size_limit`, warning instead of throw if limit set fail. [#4473 \(proller\)](#)

- Removed the `inline` tags of `void readBinary(...)` in `Field.cpp`. Also merged redundant `namespace DB` blocks. [#4530 \(hczi\)](#)

## ClickHouse Release 19.3.5, 2019-02-21

### Bug Fixes

- Fixed bug with large http insert queries processing. [#4454 \(alesapin\)](#)
- Fixed backward incompatibility with old versions due to wrong implementation of `send_logs_level` setting. [#4445 \(alexey-milovidov\)](#)
- Fixed backward incompatibility of table function `remote` introduced with column comments. [#4446 \(alexey-milovidov\)](#)

## ClickHouse Release 19.3.4, 2019-02-16

### Improvements

- Table index size is not accounted for memory limits when doing `ATTACH TABLE` query. Avoided the possibility that a table cannot be attached after being detached. [#4396 \(alexey-milovidov\)](#)
- Slightly raised up the limit on max string and array size received from ZooKeeper. It allows to continue to work with increased size of `CLIENT_JVMFLAGS=-Djute.maxbuffer=...` on ZooKeeper. [#4398 \(alexey-milovidov\)](#)
- Allow to repair abandoned replica even if it already has huge number of nodes in its queue. [#4399 \(alexey-milovidov\)](#)
- Add one required argument to `SET` index (max stored rows number). [#4386 \(Nikita Vasilev\)](#)

### Bug Fixes

- Fixed `WITH ROLLUP` result for group by single `LowCardinality` key. [#4384 \(Nikolai Kochetov\)](#)
- Fixed bug in the set index (dropping a granule if it contains more than `max_rows` rows). [#4386 \(Nikita Vasilev\)](#)
- A lot of FreeBSD build fixes. [#4397 \(proller\)](#)
- Fixed aliases substitution in queries with subquery containing same alias (issue [#4110](#)). [#4351 \(Artem Zuikov\)](#)

### Build/Testing/Packaging Improvements

- Add ability to run `clickhouse-server` for stateless tests in docker image. [#4347 \(Vasily Nemkov\)](#)

## ClickHouse Release 19.3.3, 2019-02-13

### New Features

- Added the `KILL MUTATION` statement that allows removing mutations that are for some reasons stuck. Added `latest_failed_part`, `latest_fail_time`, `latest_fail_reason` fields to the `system.mutations` table for easier troubleshooting. [#4287 \(Alex Zatelepin\)](#)
- Added aggregate function `entropy` which computes Shannon entropy. [#4238 \(Quid37\)](#)
- Added ability to send queries `INSERT INTO tbl VALUES (...)` to server without splitting on `query` and `data` parts. [#4301 \(alesapin\)](#)
- Generic implementation of `arrayWithConstant` function was added. [#4322 \(alexey-milovidov\)](#)
- Implemented `NOT BETWEEN` comparison operator. [#4228 \(Dmitry Naumov\)](#)

- Implement `sumMapFiltered` in order to be able to limit the number of keys for which values will be summed by `sumMap`. #4129 (Léo Ercolanelli)
- Added support of `Nullable` types in `mysql` table function. #4198 (Emmanuel Donin de Rosière)
- Support for arbitrary constant expressions in `LIMIT` clause. #4246 (k3box)
- Added `topKWeighted` aggregate function that takes additional argument with (unsigned integer) weight. #4245 (Andrew Golman)
- `StorageJoin` now supports `join_any_take_last_row` setting that allows overwriting existing values of the same key. #3973 (Amos Bird)
- Added function `toStartOfInterval`. #4304 (Vitaly Baranov)
- Added `RowBinaryWithNamesAndTypes` format. #4200 (Oleg V. Kozlyuk)
- Added `IPv4` and `IPv6` data types. More effective implementations of `IPv*` functions. #3669 (Vasily Nemkov)
- Added function `toStartOfTenMinutes()`. #4298 (Vitaly Baranov)
- Added `Protobuf` output format. #4005 #4158 (Vitaly Baranov)
- Added `brotli` support for HTTP interface for data import (INSERTs). #4235 (Mikhail)
- Added hints while user make typo in function name or type in command line client. #4239 (Danila Kutenin)
- Added `Query-Id` to Server's HTTP Response header. #4231 (Mikhail)

## Experimental Features

- Added `minmax` and `set` data skipping indices for MergeTree table engines family. #4143 (Nikita Vasilev)
- Added conversion of `CROSS JOIN` to `INNER JOIN` if possible. #4221 #4266 (Artem Zuikov)

## Bug Fixes

- Fixed `Not found column` for duplicate columns in `JOIN ON` section. #4279 (Artem Zuikov)
- Make `START REPLICATED SENDS` command start replicated sends. #4229 (nvartolomei)
- Fixed aggregate functions execution with `Array(LowCardinality)` arguments. #4055 (KochetovNicolai)
- Fixed wrong behaviour when doing `INSERT ... SELECT ... FROM file(...)` query and file has `CSVWithNames` or `TSVWithNames` format and the first data row is missing. #4297 (alexey-milovidov)
- Fixed crash on dictionary reload if dictionary not available. This bug was appeared in 19.1.6. #4188 (proller)
- Fixed `ALL JOIN` with duplicates in right table. #4184 (Artem Zuikov)
- Fixed segmentation fault with `use_uncompressed_cache=1` and exception with wrong uncompressed size. This bug was appeared in 19.1.6. #4186 (alesapin)
- Fixed `compile_expressions` bug with comparison of big (more than `int16`) dates. #4341 (alesapin)
- Fixed infinite loop when selecting from table function `numbers(0)`. #4280 (alexey-milovidov)
- Temporarily disable predicate optimization for `ORDER BY`. #3890 (Winter Zhang)
- Fixed `Illegal instruction` error when using `base64` functions on old CPUs. This error has been reproduced only when ClickHouse was compiled with `gcc-8`. #4275 (alexey-milovidov)

- Fixed No message received error when interacting with PostgreSQL ODBC Driver through TLS connection. Also fixes segfault when using MySQL ODBC Driver. [#4170 \(alexey-milovidov\)](#)
- Fixed incorrect result when `Date` and `DateTime` arguments are used in branches of conditional operator (function `if`). Added generic case for function `if`. [#4243 \(alexey-milovidov\)](#)
- ClickHouse dictionaries now load within `clickhouse` process. [#4166 \(alexey-milovidov\)](#)
- Fixed deadlock when `SELECT` from a table with `File` engine was retried after `No such file or directory` error. [#4161 \(alexey-milovidov\)](#)
- Fixed race condition when selecting from `system.tables` may give `table does not exist` error. [#4313 \(alexey-milovidov\)](#)
- `clickhouse-client` can segfault on exit while loading data for command line suggestions if it was run in interactive mode. [#4317 \(alexey-milovidov\)](#)
- Fixed a bug when the execution of mutations containing `IN` operators was producing incorrect results. [#4099 \(Alex Zatelepin\)](#)
- Fixed error: if there is a database with `Dictionary` engine, all dictionaries forced to load at server startup, and if there is a dictionary with ClickHouse source from localhost, the dictionary cannot load. [#4255 \(alexey-milovidov\)](#)
- Fixed error when system logs are tried to create again at server shutdown. [#4254 \(alexey-milovidov\)](#)
- Correctly return the right type and properly handle locks in `joinGet` function. [#4153 \(Amos Bird\)](#)
- Added `sumMapWithOverflow` function. [#4151 \(Léo Ercolanelli\)](#)
- Fixed segfault with `allow_experimental_multiple_joins_emulation`. [52de2c \(Artem Zuikov\)](#)
- Fixed bug with incorrect `Date` and `DateTime` comparison. [#4237 \(valexey\)](#)
- Fixed fuzz test under undefined behavior sanitizer: added parameter type check for `quantile*Weighted` family of functions. [#4145 \(alexey-milovidov\)](#)
- Fixed rare race condition when removing of old data parts can fail with `File not found` error. [#4378 \(alexey-milovidov\)](#)
- Fix install package with missing `/etc/clickhouse-server/config.xml`. [#4343 \(proller\)](#)

## Build/Testing/Packaging Improvements

- Debian package: correct `/etc/clickhouse-server/preprocessed` link according to config. [#4205 \(proller\)](#)
- Various build fixes for FreeBSD. [#4225 \(proller\)](#)
- Added ability to create, fill and drop tables in `perftest`. [#4220 \(alesapin\)](#)
- Added a script to check for duplicate includes. [#4326 \(alexey-milovidov\)](#)
- Added ability to run queries by index in performance test. [#4264 \(alesapin\)](#)
- Package with debug symbols is suggested to be installed. [#4274 \(alexey-milovidov\)](#)
- Refactoring of `performance-test`. Better logging and signals handling. [#4171 \(alesapin\)](#)
- Added docs to anonymized Yandex.Metrika datasets. [#4164 \(alesapin\)](#)
- Added tool for converting an old month-partitioned part to the custom-partitioned format. [#4195 \(Alex Zatelepin\)](#)

- Added docs about two datasets in s3. #4144 (alesapin)
- Added script which creates changelog from pull requests description. #4169 #4173 (KochetovNicolai) (KochetovNicolai)
- Added puppet module for ClickHouse. #4182 (Maxim Fedotov)
- Added docs for a group of undocumented functions. #4168 (Winter Zhang)
- ARM build fixes. #4210#4306 #4291 (proller) (proller)
- Dictionary tests now able to run from `ctest`. #4189 (proller)
- Now `/etc/ssl` is used as default directory with SSL certificates. #4167 (alexey-milovidov)
- Added checking SSE and AVX instruction at start. #4234 (lgr)
- Init script will wait server until start. #4281 (proller)

## Backward Incompatible Changes

- Removed `allow_experimental_low_cardinality_type` setting. `LowCardinality` data types are production ready. #4323 (alexey-milovidov)
- Reduce mark cache size and uncompressed cache size accordingly to available memory amount. #4240 (Lopatin Konstantin)
- Added keyword `INDEX` in `CREATE TABLE` query. A column with name `index` must be quoted with backticks or double quotes: ``index``. #4143 (Nikita Vasilev)
- `sumMap` now promote result type instead of overflow. The old `sumMap` behavior can be obtained by using `sumMapWithOverflow` function. #4151 (Léo Ercolanelli)

## Performance Improvements

- `std::sort` replaced by `pdqsort` for queries without `LIMIT`. #4236 (Evgenii Pravda)
- Now server reuse threads from global thread pool. This affects performance in some corner cases. #4150 (alexey-milovidov)

## Improvements

- Implemented AIO support for FreeBSD. #4305 (urgordeadbeef)
- `SELECT * FROM a JOIN b USING a, b` now return `a` and `b` columns only from the left table. #4141 (Artem Zuikov)
- Allow `-C` option of client to work as `-c` option. #4232 (syominsergey)
- Now option `--password` used without value requires password from stdin. #4230 (BSD\_Conqueror)
- Added highlighting of unescaped metacharacters in string literals that contain `LIKE` expressions or regexps. #4327 (alexey-milovidov)
- Added cancelling of HTTP read only queries if client socket goes away. #4213 (nvartolomei)
- Now server reports progress to keep client connections alive. #4215 (Ivan)
- Slightly better message with reason for `OPTIMIZE` query with `optimize_throw_if_noop` setting enabled. #4294 (alexey-milovidov)
- Added support of `--version` option for clickhouse server. #4251 (Lopatin Konstantin)
- Added `--help/-h` option to `clickhouse-server`. #4233 (Yuriy Baranov)

- Added support for scalar subqueries with aggregate function state result. [#4348](#) ([Nikolai Kochetov](#))
- Improved server shutdown time and ALTERs waiting time. [#4372](#) ([alexey-milovidov](#))
- Added info about the replicated\_can\_become\_leader setting to system.replicas and add logging if the replica won't try to become leader. [#4379](#) ([Alex Zatelepin](#))

## ClickHouse Release 19.1

### ClickHouse Release 19.1.14, 2019-03-14

- Fixed error Column ... queried more than once that may happen if the setting asterisk\_left\_columns\_only is set to 1 in case of using GLOBAL JOIN with SELECT \* (rare case). The issue does not exist in 19.3 and newer. [#6bac7d8d](#) ([Artem Zuikov](#))

### ClickHouse Release 19.1.13, 2019-03-12

This release contains exactly the same set of patches as 19.3.7.

### ClickHouse Release 19.1.10, 2019-03-03

This release contains exactly the same set of patches as 19.3.6.

## ClickHouse Release 19.1

### ClickHouse Release 19.1.9, 2019-02-21

#### Bug Fixes

- Fixed backward incompatibility with old versions due to wrong implementation of send\_logs\_level setting. [#4445](#) ([alexey-milovidov](#))
- Fixed backward incompatibility of table function remote introduced with column comments. [#4446](#) ([alexey-milovidov](#))

### ClickHouse Release 19.1.8, 2019-02-16

#### Bug Fixes

- Fix install package with missing /etc/clickhouse-server/config.xml. [#4343](#) ([proller](#))

## ClickHouse Release 19.1

### ClickHouse Release 19.1.7, 2019-02-15

#### Bug Fixes

- Correctly return the right type and properly handle locks in joinGet function. [#4153](#) ([Amos Bird](#))
- Fixed error when system logs are tried to create again at server shutdown. [#4254](#) ([alexey-milovidov](#))
- Fixed error: if there is a database with Dictionary engine, all dictionaries forced to load at server startup, and if there is a dictionary with ClickHouse source from localhost, the dictionary cannot load. [#4255](#) ([alexey-milovidov](#))
- Fixed a bug when the execution of mutations containing IN operators was producing incorrect results. [#4099](#) ([Alex Zatelepin](#))
- clickhouse-client can segfault on exit while loading data for command line suggestions if it was run in interactive mode. [#4317](#) ([alexey-milovidov](#))

- Fixed race condition when selecting from `system.tables` may give `table does not exist` error. [#4313](#) ([alexey-milovidov](#))
- Fixed deadlock when `SELECT` from a table with `File` engine was retried after `No such file or directory` error. [#4161](#) ([alexey-milovidov](#))
- Fixed an issue: local ClickHouse dictionaries are loaded via TCP, but should load within process. [#4166](#) ([alexey-milovidov](#))
- Fixed `No message received` error when interacting with PostgreSQL ODBC Driver through TLS connection. Also fixes segfault when using MySQL ODBC Driver. [#4170](#) ([alexey-milovidov](#))
- Temporarily disable predicate optimization for `ORDER BY`. [#3890](#) ([Winter Zhang](#))
- Fixed infinite loop when selecting from table function `numbers(0)`. [#4280](#) ([alexey-milovidov](#))
- Fixed `compile_expressions` bug with comparison of big (more than `int16`) dates. [#4341](#) ([alesapin](#))
- Fixed segmentation fault with `uncompressed_cache=1` and exception with wrong uncompressed size. [#4186](#) ([alesapin](#))
- Fixed ALL JOIN with duplicates in right table. [#4184](#) ([Artem Zuikov](#))
- Fixed wrong behaviour when doing `INSERT ... SELECT ... FROM file(...)` query and file has `CSVWithNames` or `TSVWithNames` format and the first data row is missing. [#4297](#) ([alexey-milovidov](#))
- Fixed aggregate functions execution with `Array(LowCardinality)` arguments. [#4055](#) ([KochetovNicolai](#))
- Debian package: correct `/etc/clickhouse-server/preprocessed` link according to config. [#4205](#) ([proller](#))
- Fixed fuzz test under undefined behavior sanitizer: added parameter type check for `quantile*Weighted` family of functions. [#4145](#) ([alexey-milovidov](#))
- Make `START REPLICATED SENDS` command start replicated sends. [#4229](#) ([nvartolomei](#))
- Fixed `Not found column` for duplicate columns in `JOIN ON` section. [#4279](#) ([Artem Zuikov](#))
- Now `/etc/ssl` is used as default directory with SSL certificates. [#4167](#) ([alexey-milovidov](#))
- Fixed crash on dictionary reload if dictionary not available. [#4188](#) ([proller](#))
- Fixed bug with incorrect `Date` and `DateTime` comparison. [#4237](#) ([valexey](#))
- Fixed incorrect result when `Date` and `DateTime` arguments are used in branches of conditional operator (function `if`). Added generic case for function `if`. [#4243](#) ([alexey-milovidov](#))

## ClickHouse Release 19.1.6, 2019-01-24

### New Features

- Custom per column compression codecs for tables. [#3899](#) [#4111](#) ([alesapin](#), [Winter Zhang](#), [Anatoly](#))
- Added compression codec Delta. [#4052](#) ([alesapin](#))
- Allow to `ALTER` compression codecs. [#4054](#) ([alesapin](#))
- Added functions `left`, `right`, `trim`, `ltrim`, `rtrim`, `timestampadd`, `timestampsub` for SQL standard compatibility. [#3826](#) ([Ivan Blinkov](#))
- Support for write in `HDFS` tables and `hdfs` table function. [#4084](#) ([alesapin](#))
- Added functions to search for multiple constant strings from big haystack: `multiPosition`, `multiSearch`, `firstMatch` also with `-UTF8`, `-CaseInsensitive`, and `-CaseInsensitiveUTF8` variants. [#4053](#) ([Danila Kutenin](#))

- Pruning of unused shards if `SELECT` query filters by sharding key (setting `optimize_skip_unused_shards`).  
[#3851 \(Gleb Kanterov, Ivan\)](#)
- Allow `Kafka` engine to ignore some number of parsing errors per block. [#4094 \(Ivan\)](#)
- Added support for `CatBoost` multiclass models evaluation. Function `modelEvaluate` returns tuple with per-class raw predictions for multiclass models. `libcatboostmodel.so` should be built with [#607](#). [#3959 \(KochetovNicola\)](#)
- Added functions `filesystemAvailable`, `filesystemFree`, `filesystemCapacity`. [#4097 \(Boris Granveaud\)](#)
- Added hashing functions `xxHash64` and `xxHash32`. [#3905 \(filimonov\)](#)
- Added `gccMurmurHash` hashing function (GCC flavoured Murmur hash) which uses the same hash seed as `gcc` [#4000 \(sundyli\)](#)
- Added hashing functions `javaHash`, `hiveHash`. [#3811 \(shangshujie365\)](#)
- Added table function `remoteSecure`. Function works as `remote`, but uses secure connection. [#4088 \(proller\)](#)

## Experimental Features

- Added multiple JOINs emulation (`allow_experimental_multiple_joins_emulation` setting). [#3946 \(Artem Zuikov\)](#)

## Bug Fixes

- Make `compiled_expression_cache_size` setting limited by default to lower memory consumption. [#4041 \(alesapin\)](#)
- Fix a bug that led to hangups in threads that perform ALTERs of Replicated tables and in the thread that updates configuration from ZooKeeper. [#2947](#) [#3891](#) [#3934 \(Alex Zatelepin\)](#)
- Fixed a race condition when executing a distributed ALTER task. The race condition led to more than one replica trying to execute the task and all replicas except one failing with a ZooKeeper error. [#3904 \(Alex Zatelepin\)](#)
- Fix a bug when `from_zk` config elements weren't refreshed after a request to ZooKeeper timed out. [#2947](#) [#3947 \(Alex Zatelepin\)](#)
- Fix bug with wrong prefix for IPv4 subnet masks. [#3945 \(alesapin\)](#)
- Fixed crash (`std::terminate`) in rare cases when a new thread cannot be created due to exhausted resources. [#3956 \(alexey-milovidov\)](#)
- Fix bug when in `remote` table function execution when wrong restrictions were used for in `getStructureOfRemoteTable`. [#4009 \(alesapin\)](#)
- Fix a leak of netlink sockets. They were placed in a pool where they were never deleted and new sockets were created at the start of a new thread when all current sockets were in use. [#4017 \(Alex Zatelepin\)](#)
- Fix bug with closing `/proc/self/fd` directory earlier than all fds were read from `/proc` after forking `odbc-bridge` subprocess. [#4120 \(alesapin\)](#)
- Fixed String to UInt monotonic conversion in case of usage String in primary key. [#3870 \(Winter Zhang\)](#)
- Fixed error in calculation of integer conversion function monotonicity. [#3921 \(alexey-milovidov\)](#)
- Fixed segfault in `arrayEnumerateUniq`, `arrayEnumerateDense` functions in case of some invalid arguments. [#3909 \(alexey-milovidov\)](#)
- Fix UB in `StorageMerge`. [#3910 \(Amos Bird\)](#)

- Fixed segfault in functions `addDays`, `subtractDays`. #3913 (alexey-milovidov)
- Fixed error: functions `round`, `floor`, `trunc`, `ceil` may return bogus result when executed on integer argument and large negative scale. #3914 (alexey-milovidov)
- Fixed a bug induced by ‘kill query sync’ which leads to a core dump. #3916 (muVulDeePecker)
- Fix bug with long delay after empty replication queue. #3928 #3932 (alesapin)
- Fixed excessive memory usage in case of inserting into table with `LowCardinality` primary key. #3955 (KochetovNicolai)
- Fixed `LowCardinality` serialization for `Native` format in case of empty arrays. #3907 #4011 (KochetovNicolai)
- Fixed incorrect result while using distinct by single `LowCardinality` numeric column. #3895 #4012 (KochetovNicolai)
- Fixed specialized aggregation with `LowCardinality` key (in case when `compile` setting is enabled). #3886 (KochetovNicolai)
- Fix user and password forwarding for replicated tables queries. #3957 (alesapin) (小路)
- Fixed very rare race condition that can happen when listing tables in Dictionary database while reloading dictionaries. #3970 (alexey-milovidov)
- Fixed incorrect result when HAVING was used with ROLLUP or CUBE. #3756 #3837 (Sam Chou)
- Fixed column aliases for query with `JOIN ON` syntax and distributed tables. #3980 (Winter Zhang)
- Fixed error in internal implementation of `quantileTDigest` (found by Artem Vakhrushev). This error never happens in ClickHouse and was relevant only for those who use ClickHouse codebase as a library directly. #3935 (alexey-milovidov)

## Improvements

- Support for `IF NOT EXISTS` in `ALTER TABLE ADD COLUMN` statements along with `IF EXISTS` in `DROP/MODIFY/CLEAR/COMMENT COLUMN`. #3900 (Boris Granveaud)
- Function `parseDateTimeBestEffort`: support for formats `DD.MM.YYYY`, `DD.MM.YY`, `DD-MM-YYYY`, `DD-Mon-YYYY`, `DD/Month/YYYY` and similar. #3922 (alexey-milovidov)
- `CapnProtoInputStream` now support jagged structures. #4063 (Odin Hultgren Van Der Horst)
- Usability improvement: added a check that server process is started from the data directory’s owner. Do not allow to start server from root if the data belongs to non-root user. #3785 (sergey-v-galtsev)
- Better logic of checking required columns during analysis of queries with JOINs. #3930 (Artem Zuikov)
- Decreased the number of connections in case of large number of Distributed tables in a single server. #3726 (Winter Zhang)
- Supported totals row for `WITH TOTALS` query for ODBC driver. #3836 (Maksim Koritckiy)
- Allowed to use `Enums` as integers inside if function. #3875 (Ivan)
- Added `low_cardinality_allow_in_native_format` setting. If disabled, do not use `LowCardinality` type in `Native` format. #3879 (KochetovNicolai)
- Removed some redundant objects from compiled expressions cache to lower memory usage. #4042 (alesapin)

- Add check that `SET send_logs_level = 'value'` query accept appropriate value. #3873 (Sabyanin Maxim)
- Fixed data type check in type conversion functions. #3896 (Winter Zhang)

## Performance Improvements

- Add a MergeTree setting `use_minimalistic_part_header_in_zookeeper`. If enabled, Replicated tables will store compact part metadata in a single part znode. This can dramatically reduce ZooKeeper snapshot size (especially if the tables have a lot of columns). Note that after enabling this setting you will not be able to downgrade to a version that does not support it. #3960 (Alex Zatelepin)
- Add an DFA-based implementation for functions `sequenceMatch` and `sequenceCount` in case pattern does not contain time. #4004 (Léo Ercolanelli)
- Performance improvement for integer numbers serialization. #3968 (Amos Bird)
- Zero left padding `PODArray` so that -1 element is always valid and zeroed. It's used for branchless calculation of offsets. #3920 (Amos Bird)
- Reverted `jemalloc` version which lead to performance degradation. #4018 (alexey-milovidov)

## Backward Incompatible Changes

- Removed undocumented feature `ALTER MODIFY PRIMARY KEY` because it was superseded by the `ALTER MODIFY ORDER BY` command. #3887 (Alex Zatelepin)
- Removed function `shardByHash`. #3833 (alexey-milovidov)
- Forbid using scalar subqueries with result of type `AggregateFunction`. #3865 (Ivan)

## Build/Testing/Packaging Improvements

- Added support for PowerPC (`ppc64le`) build. #4132 (Danila Kutenin)
- Stateful functional tests are run on public available dataset. #3969 (alexey-milovidov)
- Fixed error when the server cannot start with the `bash: /usr/bin/clickhouse-extract-from-config: Operation not permitted` message within Docker or `systemd-nspawn`. #4136 (alexey-milovidov)
- Updated `rdkafka` library to v1.0.0-RC5. Used `cppkafka` instead of raw C interface. #4025 (Ivan)
- Updated `mariadb-client` library. Fixed one of issues found by UBSan. #3924 (alexey-milovidov)
- Some fixes for UBSan builds. #3926 #3021 #3948 (alexey-milovidov)
- Added per-commit runs of tests with UBSan build.
- Added per-commit runs of PVS-Studio static analyzer.
- Fixed bugs found by PVS-Studio. #4013 (alexey-milovidov)
- Fixed glibc compatibility issues. #4100 (alexey-milovidov)
- Move Docker images to 18.10 and add compatibility file for glibc >= 2.28 #3965 (alesapin)
- Add env variable if user do not want to chown directories in server Docker image. #3967 (alesapin)
- Enabled most of the warnings from `-Weverything` in clang. Enabled `-Wpedantic`. #3986 (alexey-milovidov)
- Added a few more warnings that are available only in clang 8. #3993 (alexey-milovidov)
- Link to `libLLVM` rather than to individual LLVM libs when using shared linking. #3989 (Orivej Desh)
- Added sanitizer variables for test images. #4072 (alesapin)

- clickhouse-server debian package will recommend `libcap2-bin` package to use `setcap` tool for setting capabilities. This is optional. [#4093 \(alexey-milovidov\)](#)
- Improved compilation time, fixed includes. [#3898 \(proller\)](#)
- Added performance tests for hash functions. [#3918 \(filimonov\)](#)
- Fixed cyclic library dependences. [#3958 \(proller\)](#)
- Improved compilation with low available memory. [#4030 \(proller\)](#)
- Added test script to reproduce performance degradation in jemalloc. [#4036 \(alexey-milovidov\)](#)
- Fixed misspells in comments and string literals under dbms. [#4122 \(maiha\)](#)
- Fixed typos in comments. [#4089 \(Evgenii Pravda\)](#)

## Changelog for 2018

---

### ClickHouse Release 18.16

#### ClickHouse Release 18.16.1, 2018-12-21

##### Bug Fixes:

- Fixed an error that led to problems with updating dictionaries with the ODBC source. [#3825, #3829](#)
- JIT compilation of aggregate functions now works with LowCardinality columns. [#3838](#)

##### Improvements:

- Added the `low_cardinality_allow_in_native_format` setting (enabled by default). When disabled, LowCardinality columns will be converted to ordinary columns for SELECT queries and ordinary columns will be expected for INSERT queries. [#3879](#)

##### Build Improvements:

- Fixes for builds on macOS and ARM.

#### ClickHouse Release 18.16.0, 2018-12-14

##### New Features:

- DEFAULT expressions are evaluated for missing fields when loading data in semi-structured input formats (`JSONEachRow`, `TSKV`). The feature is enabled with the `insert_sample_with_metadata` setting. [#3555](#)
- The `ALTER TABLE` query now has the `MODIFY ORDER BY` action for changing the sorting key when adding or removing a table column. This is useful for tables in the `MergeTree` family that perform additional tasks when merging based on this sorting key, such as `SummingMergeTree`, `AggregatingMergeTree`, and so on. [#3581 #3755](#)
- For tables in the `MergeTree` family, now you can specify a different sorting key (`ORDER BY`) and index (`PRIMARY KEY`). The sorting key can be longer than the index. [#3581](#)
- Added the `hdfs` table function and the `HDFS` table engine for importing and exporting data to HDFS. [chenxing-xc](#)
- Added functions for working with base64: `base64Encode`, `base64Decode`, `tryBase64Decode`. [Alexander Krasheninnikov](#)

- Now you can use a parameter to configure the precision of the `uniqCombined` aggregate function (select the number of HyperLogLog cells). [#3406](#)
- Added the `system.contributors` table that contains the names of everyone who made commits in ClickHouse. [#3452](#)
- Added the ability to omit the partition for the `ALTER TABLE ... FREEZE` query in order to back up all partitions at once. [#3514](#)
- Added `dictGet` and `dictGetOrDefault` functions that do not require specifying the type of return value. The type is determined automatically from the dictionary description. [Amos Bird](#)
- Now you can specify comments for a column in the table description and change it using `ALTER`. [#3377](#)
- Reading is supported for `Join` type tables with simple keys. [Amos Bird](#)
- Now you can specify the options `join_use_nulls`, `max_rows_in_join`, `max_bytes_in_join`, and `join_overflow_mode` when creating a `Join` type table. [Amos Bird](#)
- Added the `joinGet` function that allows you to use a `Join` type table like a dictionary. [Amos Bird](#)
- Added the `partition_key`, `sorting_key`, `primary_key`, and `sampling_key` columns to the `system.tables` table in order to provide information about table keys. [#3609](#)
- Added the `is_in_partition_key`, `is_in_sorting_key`, `is_in_primary_key`, and `is_in_sampling_key` columns to the `system.columns` table. [#3609](#)
- Added the `min_time` and `max_time` columns to the `system.parts` table. These columns are populated when the partitioning key is an expression consisting of `DateTime` columns. [Emmanuel Donin de Rosière](#)

## Bug Fixes:

- Fixes and performance improvements for the `LowCardinality` data type. `GROUP BY` using `LowCardinality(Nullable(...))`. Getting the values of extremes. Processing high-order functions. `LEFT ARRAY JOIN`. Distributed `GROUP BY`. Functions that return `Array`. Execution of `ORDER BY`. Writing to `Distributed` tables (nicelulu). Backward compatibility for `INSERT` queries from old clients that implement the `Native` protocol. Support for `LowCardinality` for `JOIN`. Improved performance when working in a single stream. [#3823](#) [#3803](#) [#3799](#) [#3769](#) [#3744](#) [#3681](#) [#3651](#) [#3649](#) [#3641](#) [#3632](#) [#3568](#) [#3523](#) [#3518](#)
- Fixed how the `select_sequential_consistency` option works. Previously, when this setting was enabled, an incomplete result was sometimes returned after beginning to write to a new partition. [#2863](#)
- Databases are correctly specified when executing DDL `ON CLUSTER` queries and `ALTER UPDATE/DELETE`. [#3772](#) [#3460](#)
- Databases are correctly specified for subqueries inside a `VIEW`. [#3521](#)
- Fixed a bug in `PREWHERE` with `FINAL` for `VersionedCollapsingMergeTree`. [7167bfd7](#)
- Now you can use `KILL QUERY` to cancel queries that have not started yet because they are waiting for the table to be locked. [#3517](#)
- Corrected date and time calculations if the clocks were moved back at midnight (this happens in Iran, and happened in Moscow from 1981 to 1983). Previously, this led to the time being reset a day earlier than necessary, and also caused incorrect formatting of the date and time in text format. [#3819](#)
- Fixed bugs in some cases of `VIEW` and subqueries that omit the database. [Winter Zhang](#)
- Fixed a race condition when simultaneously reading from a `MATERIALIZED VIEW` and deleting a `MATERIALIZED VIEW` due to not locking the internal `MATERIALIZED VIEW`. [#3404](#) [#3694](#)

- Fixed the error `Lock handler cannot be nullptr`. [#3689](#)
- Fixed query processing when the `compile_expressions` option is enabled (it's enabled by default). Nondeterministic constant expressions like the `now` function are no longer unfolded. [#3457](#)
- Fixed a crash when specifying a non-constant scale argument in `toDecimal32/64/128` functions.
- Fixed an error when trying to insert an array with `NULL` elements in the `Values` format into a column of type `Array` without `Nullable` (if `input_format_values_interpret_expressions = 1`). [#3487](#) [#3503](#)
- Fixed continuous error logging in `DDLWorker` if ZooKeeper is not available. [8f50c620](#)
- Fixed the return type for `quantile*` functions from `Date` and `DateTime` types of arguments. [#3580](#)
- Fixed the `WITH` clause if it specifies a simple alias without expressions. [#3570](#)
- Fixed processing of queries with named sub-queries and qualified column names when `enable_optimize_predicate_expression` is enabled. [Winter Zhang](#)
- Fixed the error `Attempt to attach to nullptr thread group` when working with materialized views. [Marek Vavruša](#)
- Fixed a crash when passing certain incorrect arguments to the `arrayReverse` function. [73e3a7b6](#)
- Fixed the buffer overflow in the `extractURLParameter` function. Improved performance. Added correct processing of strings containing zero bytes. [141e9799](#)
- Fixed buffer overflow in the `lowerUTF8` and `upperUTF8` functions. Removed the ability to execute these functions over `FixedString` type arguments. [#3662](#)
- Fixed a rare race condition when deleting `MergeTree` tables. [#3680](#)
- Fixed a race condition when reading from `Buffer` tables and simultaneously performing `ALTER` or `DROP` on the target tables. [#3719](#)
- Fixed a segfault if the `max_temporary_non_const_columns` limit was exceeded. [#3788](#)

## Improvements:

- The server does not write the processed configuration files to the `/etc/clickhouse-server/` directory. Instead, it saves them in the `preprocessed_configs` directory inside `path`. This means that the `/etc/clickhouse-server/` directory does not have write access for the `clickhouse` user, which improves security. [#2443](#)
- The `min_merge_bytes_to_use_direct_io` option is set to 10 GiB by default. A merge that forms large parts of tables from the `MergeTree` family will be performed in `O_DIRECT` mode, which prevents excessive page cache eviction. [#3504](#)
- Accelerated server start when there is a very large number of tables. [#3398](#)
- Added a connection pool and HTTP Keep-Alive for connections between replicas. [#3594](#)
- If the query syntax is invalid, the 400 Bad Request code is returned in the `HTTP` interface (500 was returned previously). [31bc680a](#)
- The `join_default_strictness` option is set to `ALL` by default for compatibility. [120e2cbe](#)
- Removed logging to `stderr` from the `re2` library for invalid or complex regular expressions. [#3723](#)
- Added for the `Kafka` table engine: checks for subscriptions before beginning to read from Kafka; the `kafka_max_block_size` setting for the table. [Marek Vavruša](#)

- The `cityHash64`, `farmHash64`, `metroHash64`, `sipHash64`, `halfMD5`, `murmurHash2_32`, `murmurHash2_64`, `murmurHash3_32`, and `murmurHash3_64` functions now work for any number of arguments and for arguments in the form of tuples. [#3451](#) [#3519](#)
- The `arrayReverse` function now works with any types of arrays. [#73e3a7b6](#)
- Added an optional parameter: the slot size for the `timeSlots` function. [Kirill Shvakov](#)
- For `FULL` and `RIGHT JOIN`, the `max_block_size` setting is used for a stream of non-joined data from the right table. [Amos Bird](#)
- Added the `--secure` command line parameter in `clickhouse-benchmark` and `clickhouse-performance-test` to enable TLS. [#3688](#) [#3690](#)
- Type conversion when the structure of a `Buffer` type table does not match the structure of the destination table. [Vitaly Baranov](#)
- Added the `tcp_keep_alive_timeout` option to enable keep-alive packets after inactivity for the specified time interval. [#3441](#)
- Removed unnecessary quoting of values for the partition key in the `system.parts` table if it consists of a single column. [#3652](#)
- The modulo function works for `Date` and `DateTime` data types. [#3385](#)
- Added synonyms for the `POWER`, `LN`, `LCASE`, `UCASE`, `REPLACE`, `LOCATE`, `SUBSTR`, and `MID` functions. [#3774](#) [#3763](#) Some function names are case-insensitive for compatibility with the SQL standard. Added syntactic sugar `SUBSTRING(expr FROM start FOR length)` for compatibility with SQL. [#3804](#)
- Added the ability to mlock memory pages corresponding to `clickhouse-server` executable code to prevent it from being forced out of memory. This feature is disabled by default. [#3553](#)
- Improved performance when reading from `O_DIRECT` (with the `min_bytes_to_use_direct_io` option enabled). [#3405](#)
- Improved performance of the `dictGet...OrDefault` function for a constant key argument and a non-constant default argument. [Amos Bird](#)
- The `firstSignificantSubdomain` function now processes the domains `gov`, `mil`, and `edu`. [Igor Hatarist](#) Improved performance. [#3628](#)
- Ability to specify custom environment variables for starting `clickhouse-server` using the `SYS-V init.d` script by defining `CLICKHOUSE_PROGRAM_ENV` in `/etc/default/clickhouse`.  
[Pavlo Bashynskyi](#)
- Correct return code for the `clickhouse-server` init script. [#3516](#)
- The `system.metrics` table now has the `VersionInteger` metric, and `system.build_options` has the added line `VERSION_INTEGER`, which contains the numeric form of the ClickHouse version, such as `18016000`. [#3644](#)
- Removed the ability to compare the `Date` type with a number to avoid potential errors like `date = 2018-12-17`, where quotes around the date are omitted by mistake. [#3687](#)
- Fixed the behavior of stateful functions like `rowNumberInAllBlocks`. They previously output a result that was one number larger due to starting during query analysis. [Amos Bird](#)
- If the `force_restore_data` file can't be deleted, an error message is displayed. [Amos Bird](#)

## Build Improvements:

- Updated the `jemalloc` library, which fixes a potential memory leak. [Amos Bird](#)

- Profiling with `jemalloc` is enabled by default in order to debug builds. [#2cc82f5c](#)
- Added the ability to run integration tests when only `Docker` is installed on the system. [#3650](#)
- Added the fuzz expression test in `SELECT` queries. [#3442](#)
- Added a stress test for commits, which performs functional tests in parallel and in random order to detect more race conditions. [#3438](#)
- Improved the method for starting `clickhouse-server` in a Docker image. [Elghazal Ahmed](#)
- For a Docker image, added support for initializing databases using files in the `/docker-entrypoint-initdb.d` directory. [Konstantin Lebedev](#)
- Fixes for builds on ARM. [#3709](#)

#### Backward Incompatible Changes:

- Removed the ability to compare the `Date` type with a number. Instead of `toDate('2018-12-18') = 17883`, you must use explicit type conversion `= toDate(17883)` [#3687](#)

## ClickHouse Release 18.14

### ClickHouse Release 18.14.19, 2018-12-19

#### Bug Fixes:

- Fixed an error that led to problems with updating dictionaries with the ODBC source. [#3825](#), [#3829](#)
- Databases are correctly specified when executing DDL `ON CLUSTER` queries. [#3460](#)
- Fixed a segfault if the `max_temporary_non_const_columns` limit was exceeded. [#3788](#)

#### Build Improvements:

- Fixes for builds on ARM.

### ClickHouse Release 18.14.18, 2018-12-04

#### Bug Fixes:

- Fixed error in `dictGet...` function for dictionaries of type `range`, if one of the arguments is constant and other is not. [#3751](#)
- Fixed error that caused messages `netlink: ...: attribute type 1 has an invalid length` to be printed in Linux kernel log, that was happening only on fresh enough versions of Linux kernel. [#3749](#)
- Fixed segfault in function `empty` for argument of `FixedString` type. [Daniel, Dao Quang Minh](#)
- Fixed excessive memory allocation when using large value of `max_query_size` setting (a memory chunk of `max_query_size` bytes was preallocated at once). [#3720](#)

#### Build Changes:

- Fixed build with LLVM/Clang libraries of version 7 from the OS packages (these libraries are used for runtime query compilation). [#3582](#)

### ClickHouse Release 18.14.17, 2018-11-30

#### Bug Fixes:

- Fixed cases when the ODBC bridge process did not terminate with the main server process. [#3642](#)

- Fixed synchronous insertion into the `Distributed` table with a columns list that differs from the column list of the remote table. [#3673](#)
- Fixed a rare race condition that can lead to a crash when dropping a MergeTree table. [#3643](#)
- Fixed a query deadlock in case when query thread creation fails with the `Resource temporarily unavailable` error. [#3643](#)
- Fixed parsing of the `ENGINE` clause when the `CREATE AS table` syntax was used and the `ENGINE` clause was specified before the `AS table` (the error resulted in ignoring the specified engine). [#3692](#)

## ClickHouse Release 18.14.15, 2018-11-21

### Bug Fixes:

- The size of memory chunk was overestimated while deserializing the column of type `Array(String)` that leads to “Memory limit exceeded” errors. The issue appeared in version 18.12.13. [#3589](#)

## ClickHouse Release 18.14.14, 2018-11-20

### Bug Fixes:

- Fixed `ON CLUSTER` queries when cluster configured as secure (flag `<secure>`). [#3599](#)

### Build Changes:

- Fixed problems (Ilvm-7 from system, macos) [#3582](#)

## ClickHouse Release 18.14.13, 2018-11-08

### Bug Fixes:

- Fixed the `Block` structure mismatch in `MergingSorted stream` error. [#3162](#)
- Fixed `ON CLUSTER` queries in case when secure connections were turned on in the cluster config (the `<secure>` flag). [#3465](#)
- Fixed an error in queries that used `SAMPLE`, `PREWHERE` and alias columns. [#3543](#)
- Fixed a rare `unknown compression method` error when the `min_bytes_to_use_direct_io` setting was enabled. [#3544](#)

### Performance Improvements:

- Fixed performance regression of queries with `GROUP BY` of columns of `UInt16` or `Date` type when executing on AMD EPYC processors. [Igor Lapko](#)
- Fixed performance regression of queries that process long strings. [#3530](#)

### Build Improvements:

- Improvements for simplifying the Arcadia build. [#3475](#), [#3535](#)

## ClickHouse Release 18.14.12, 2018-11-02

### Bug Fixes:

- Fixed a crash on joining two unnamed subqueries. [#3505](#)
- Fixed generating incorrect queries (with an empty `WHERE` clause) when querying external databases. [hotid](#)
- Fixed using an incorrect timeout value in ODBC dictionaries. [Marek Vavruša](#)

# ClickHouse Release 18.14.11, 2018-10-29

## Bug Fixes:

- Fixed the error Block structure mismatch in UNION stream: different number of columns in LIMIT queries. [#2156](#)
- Fixed errors when merging data in tables containing arrays inside Nested structures. [#3397](#)
- Fixed incorrect query results if the merge\_tree\_uniform\_read\_distribution setting is disabled (it is enabled by default). [#3429](#)
- Fixed an error on inserts to a Distributed table in Native format. [#3411](#)

# ClickHouse Release 18.14.10, 2018-10-23

- The `compile_expressions` setting (JIT compilation of expressions) is disabled by default. [#3410](#)
- The `enable_optimize_predicate_expression` setting is disabled by default.

# ClickHouse Release 18.14.9, 2018-10-16

## New Features:

- The `WITH CUBE` modifier for `GROUP BY` (the alternative syntax `GROUP BY CUBE(...)` is also available). [#3172](#)
- Added the `formatDateTime` function. [Alexandr Krasheninnikov](#)
- Added the `JDBC` table engine and `jdbc` table function (requires installing `clickhouse-jdbc-bridge`). [Alexandr Krasheninnikov](#)
- Added functions for working with the ISO week number: `toISOWeek`, `toISOYear`, `toStartOfISOYear`, and `toDayOfYear`. [#3146](#)
- Now you can use `Nullable` columns for `MySQL` and `ODBC` tables. [#3362](#)
- Nested data structures can be read as nested objects in `JSONEachRow` format. Added the `input_format_import_nested_json` setting. [Veloman Yunkan](#)
- Parallel processing is available for many `MATERIALIZED VIEWS` when inserting data. See the `parallel_view_processing` setting. [Marek Vavruša](#)
- Added the `SYSTEM FLUSH LOGS` query (forced log flushes to system tables such as `query_log`) [#3321](#)
- Now you can use pre-defined `database` and `table` macros when declaring `Replicated` tables. [#3251](#)
- Added the ability to read `Decimal` type values in engineering notation (indicating powers of ten). [#3153](#)

## Experimental Features:

- Optimization of the `GROUP BY` clause for `LowCardinality` data types. [#3138](#)
- Optimized calculation of expressions for `LowCardinality` data types. [#3200](#)

## Improvements:

- Significantly reduced memory consumption for queries with `ORDER BY` and `LIMIT`. See the `max_bytes_before_remerge_sort` setting. [#3205](#)
- In the absence of `JOIN` (`LEFT`, `INNER`, ...), `INNER JOIN` is assumed. [#3147](#)
- Qualified asterisks work correctly in queries with `JOIN`. [Winter Zhang](#)
- The `ODBC` table engine correctly chooses the method for quoting identifiers in the SQL dialect of a remote database. [Alexandr Krasheninnikov](#)

- The `compile_expressions` setting (JIT compilation of expressions) is enabled by default.
- Fixed behavior for simultaneous `DROP DATABASE/TABLE IF EXISTS` and `CREATE DATABASE/TABLE IF NOT EXISTS`. Previously, a `CREATE DATABASE ... IF NOT EXISTS` query could return the error message “File ... already exists”, and the `CREATE TABLE ... IF NOT EXISTS` and `DROP TABLE IF EXISTS` queries could return `Table ... is creating or attaching right now.` [#3101](#)
- LIKE and IN expressions with a constant right half are passed to the remote server when querying from MySQL or ODBC tables. [#3182](#)
- Comparisons with constant expressions in a WHERE clause are passed to the remote server when querying from MySQL and ODBC tables. Previously, only comparisons with constants were passed. [#3182](#)
- Correct calculation of row width in the terminal for `Pretty` formats, including strings with hieroglyphs. [Amos Bird](#).
- `ON CLUSTER` can be specified for `ALTER UPDATE` queries.
- Improved performance for reading data in `JSONEachRow` format. [#3332](#)
- Added synonyms for the `LENGTH` and `CHARACTER_LENGTH` functions for compatibility. The `CONCAT` function is no longer case-sensitive. [#3306](#)
- Added the `TIMESTAMP` synonym for the `DateTime` type. [#3390](#)
- There is always space reserved for `query_id` in the server logs, even if the log line is not related to a query. This makes it easier to parse server text logs with third-party tools.
- Memory consumption by a query is logged when it exceeds the next level of an integer number of gigabytes. [#3205](#)
- Added compatibility mode for the case when the client library that uses the Native protocol sends fewer columns by mistake than the server expects for the `INSERT` query. This scenario was possible when using the `clickhouse-cpp` library. Previously, this scenario caused the server to crash. [#3171](#)
- In a user-defined WHERE expression in `clickhouse-copier`, you can now use a `partition_key` alias (for additional filtering by source table partition). This is useful if the partitioning scheme changes during copying, but only changes slightly. [#3166](#)
- The workflow of the `Kafka` engine has been moved to a background thread pool in order to automatically reduce the speed of data reading at high loads. [Marek Vavruša](#).
- Support for reading `Tuple` and `Nested` values of structures like `struct` in the `Cap'n'Proto` format. [Marek Vavruša](#)
- The list of top-level domains for the `firstSignificantSubdomain` function now includes the domain `biz`. [decaseal](#)
- In the configuration of external dictionaries, `null_value` is interpreted as the value of the default data type. [#3330](#)
- Support for the `intDiv` and `intDivOrZero` functions for `Decimal`. [b48402e8](#)
- Support for the `Date`, `DateTime`, `UUID`, and `Decimal` types as a key for the `sumMap` aggregate function. [#3281](#)
- Support for the `Decimal` data type in external dictionaries. [#3324](#)
- Support for the `Decimal` data type in `SummingMergeTree` tables. [#3348](#)

- Added specializations for `UUID` in `if`. [#3366](#)
- Reduced the number of `open` and `close` system calls when reading from a `MergeTree` table. [#3283](#)
- A `TRUNCATE TABLE` query can be executed on any replica (the query is passed to the leader replica). [Kirill Shvakov](#)

## Bug Fixes:

- Fixed an issue with `Dictionary` tables for `range_hashed` dictionaries. This error occurred in version 18.12.17. [#1702](#)
- Fixed an error when loading `range_hashed` dictionaries (the message `Unsupported type Nullable (...)`). This error occurred in version 18.12.17. [#3362](#)
- Fixed errors in the `pointInPolygon` function due to the accumulation of inaccurate calculations for polygons with a large number of vertices located close to each other. [#3331](#) [#3341](#)
- If after merging data parts, the checksum for the resulting part differs from the result of the same merge in another replica, the result of the merge is deleted and the data part is downloaded from the other replica (this is the correct behavior). But after downloading the data part, it couldn't be added to the working set because of an error that the part already exists (because the data part was deleted with some delay after the merge). This led to cyclical attempts to download the same data. [#3194](#)
- Fixed incorrect calculation of total memory consumption by queries (because of incorrect calculation, the `max_memory_usage_for_all_queries` setting worked incorrectly and the `MemoryTracking` metric had an incorrect value). This error occurred in version 18.12.13. [Marek Vavruša](#)
- Fixed the functionality of `CREATE TABLE ... ON CLUSTER ... AS SELECT ...`. This error occurred in version 18.12.13. [#3247](#)
- Fixed unnecessary preparation of data structures for `JOINS` on the server that initiates the query if the `JOIN` is only performed on remote servers. [#3340](#)
- Fixed bugs in the `Kafka` engine: deadlocks after exceptions when starting to read data, and locks upon completion [Marek Vavruša](#).
- For `Kafka` tables, the optional `schema` parameter was not passed (the schema of the `Cap'n'Proto` format). [Vojtech Splichal](#)
- If the ensemble of ZooKeeper servers has servers that accept the connection but then immediately close it instead of responding to the handshake, ClickHouse chooses to connect another server. Previously, this produced the error `Cannot read all data. Bytes read: 0. Bytes expected: 4.` and the server couldn't start. [8218cf3a](#)
- If the ensemble of ZooKeeper servers contains servers for which the DNS query returns an error, these servers are ignored. [17b8e209](#)
- Fixed type conversion between `Date` and `DateTime` when inserting data in the `VALUES` format (if `input_format_values_interpret_expressions = 1`). Previously, the conversion was performed between the numerical value of the number of days in Unix Epoch time and the Unix timestamp, which led to unexpected results. [#3229](#)
- Corrected type conversion between `Decimal` and integer numbers. [#3211](#)
- Fixed errors in the `enable_optimize_predicate_expression` setting. [Winter Zhang](#)
- Fixed a parsing error in CSV format with floating-point numbers if a non-default CSV separator is used, such as ; [#3155](#)

- Fixed the `arrayCumSumNonNegative` function (it does not accumulate negative values if the accumulator is less than zero). [Aleksey Studnev](#)
- Fixed how `Merge` tables work on top of `Distributed` tables when using `PREWHERE`. [#3165](#)
- Bug fixes in the `ALTER UPDATE` query.
- Fixed bugs in the `odbc` table function that appeared in version 18.12. [#3197](#)
- Fixed the operation of aggregate functions with `StateArray` combinators. [#3188](#)
- Fixed a crash when dividing a `Decimal` value by zero. [69dd6609](#)
- Fixed output of types for operations using `Decimal` and integer arguments. [#3224](#)
- Fixed the segfault during `GROUP BY` on `Decimal128`. [3359ba06](#)
- The `log_query_threads` setting (logging information about each thread of query execution) now takes effect only if the `log_queries` option (logging information about queries) is set to 1. Since the `log_query_threads` option is enabled by default, information about threads was previously logged even if query logging was disabled. [#3241](#)
- Fixed an error in the distributed operation of the quantiles aggregate function (the error message `Not found column quantile...`). [292a8855](#)
- Fixed the compatibility problem when working on a cluster of version 18.12.17 servers and older servers at the same time. For distributed queries with `GROUP BY` keys of both fixed and non-fixed length, if there was a large amount of data to aggregate, the returned data was not always fully aggregated (two different rows contained the same aggregation keys). [#3254](#)
- Fixed handling of substitutions in `clickhouse-performance-test`, if the query contains only part of the substitutions declared in the test. [#3263](#)
- Fixed an error when using `FINAL` with `PREWHERE`. [#3298](#)
- Fixed an error when using `PREWHERE` over columns that were added during `ALTER`. [#3298](#)
- Added a check for the absence of `arrayJoin` for `DEFAULT` and `MATERIALIZED` expressions. Previously, `arrayJoin` led to an error when inserting data. [#3337](#)
- Added a check for the absence of `arrayJoin` in a `PREWHERE` clause. Previously, this led to messages like `Size ... does not match` or `Unknown compression method` when executing queries. [#3357](#)
- Fixed segfault that could occur in rare cases after optimization that replaced AND chains from equality evaluations with the corresponding IN expression. [liuyimin-bytedance](#)
- Minor corrections to `clickhouse-benchmark`: previously, client information was not sent to the server; now the number of queries executed is calculated more accurately when shutting down and for limiting the number of iterations. [#3351](#) [#3352](#)

#### Backward Incompatible Changes:

- Removed the `allow_experimental_decimal_type` option. The `Decimal` data type is available for default use. [#3329](#)

## ClickHouse Release 18.12

ClickHouse Release 18.12.17, 2018-09-16

New Features:

- `invalidate_query` (the ability to specify a query to check whether an external dictionary needs to be updated) is implemented for the `clickhouse` source. [#3126](#)
- Added the ability to use `UInt*`, `Int*`, and `DateTime` data types (along with the `Date` type) as a `range_hashed` external dictionary key that defines the boundaries of ranges. Now `NULL` can be used to designate an open range. [Vasily Nemkov](#)
- The `Decimal` type now supports `var*` and `stddev*` aggregate functions. [#3129](#)
- The `Decimal` type now supports mathematical functions (`exp`, `sin` and so on.) [#3129](#)
- The `system.part_log` table now has the `partition_id` column. [#3089](#)

#### Bug Fixes:

- Merge now works correctly on `Distributed` tables. [Winter Zhang](#)
- Fixed incompatibility (unnecessary dependency on the `glibc` version) that made it impossible to run ClickHouse on `Ubuntu Precise` and older versions. The incompatibility arose in version 18.12.13. [#3130](#)
- Fixed errors in the `enable_optimize_predicate_expression` setting. [Winter Zhang](#)
- Fixed a minor issue with backwards compatibility that appeared when working with a cluster of replicas on versions earlier than 18.12.13 and simultaneously creating a new replica of a table on a server with a newer version (shown in the message `Can not clone replica, because the ...` updated to new ClickHouse version which is logical, but shouldn't happen). [#3122](#)

#### Backward Incompatible Changes:

- The `enable_optimize_predicate_expression` option is enabled by default (which is rather optimistic). If query analysis errors occur that are related to searching for the column names, set `enable_optimize_predicate_expression` to 0. [Winter Zhang](#)

## ClickHouse Release 18.12.14, 2018-09-13

#### New Features:

- Added support for `ALTER UPDATE` queries. [#3035](#)
- Added the `allow_ddl` option, which restricts the user's access to DDL queries. [#3104](#)
- Added the `min_merge_bytes_to_use_direct_io` option for `MergeTree` engines, which allows you to set a threshold for the total size of the merge (when above the threshold, data part files will be handled using `O_DIRECT`). [#3117](#)
- The `system.merges` system table now contains the `partition_id` column. [#3099](#)

#### Improvements

- If a data part remains unchanged during mutation, it isn't downloaded by replicas. [#3103](#)
- Autocomplete is available for names of settings when working with `clickhouse-client`. [#3106](#)

#### Bug Fixes:

- Added a check for the sizes of arrays that are elements of `Nested` type fields when inserting. [#3118](#)
- Fixed an error updating external dictionaries with the `ODBC` source and `hashed` storage. This error occurred in version 18.12.13.
- Fixed a crash when creating a temporary table from a query with an `IN` condition. [Winter Zhang](#)
- Fixed an error in aggregate functions for arrays that can have `NULL` elements. [Winter Zhang](#)

# ClickHouse Release 18.12.13, 2018-09-10

## New Features:

- Added the `DECIMAL(digits, scale)` data type (`Decimal32(scale)`, `Decimal64(scale)`, `Decimal128(scale)`). To enable it, use the setting `allow_experimental_decimal_type`. [#2846](#) [#2970](#) [#3008](#) [#3047](#)
- New `WITH ROLLUP` modifier for `GROUP BY` (alternative syntax: `GROUP BY ROLLUP(...)`). [#2948](#)
- In queries with `JOIN`, the star character expands to a list of columns in all tables, in compliance with the SQL standard. You can restore the old behavior by setting `asterisk_left_columns_only` to 1 on the user configuration level. [Winter Zhang](#)
- Added support for `JOIN` with table functions. [Winter Zhang](#)
- Autocomplete by pressing Tab in clickhouse-client. [Sergey Shcherbin](#)
- Ctrl+C in clickhouse-client clears a query that was entered. [#2877](#)
- Added the `join_default_strictness` setting (values: `"`, `'any'`, `'all'`). This allows you to not specify `ANY` or `ALL` for `JOIN`. [#2982](#)
- Each line of the server log related to query processing shows the query ID. [#2482](#)
- Now you can get query execution logs in clickhouse-client (use the `send_logs_level` setting). With distributed query processing, logs are cascaded from all the servers. [#2482](#)
- The `system.query_log` and `system.processes` (`SHOW PROCESSLIST`) tables now have information about all changed settings when you run a query (the nested structure of the `Settings` data). Added the `log_query_settings` setting. [#2482](#)
- The `system.query_log` and `system.processes` tables now show information about the number of threads that are participating in query execution (see the `thread_numbers` column). [#2482](#)
- Added `ProfileEvents` counters that measure the time spent on reading and writing over the network and reading and writing to disk, the number of network errors, and the time spent waiting when network bandwidth is limited. [#2482](#)
- Added `ProfileEvents` counters that contain the system metrics from `rusage` (you can use them to get information about CPU usage in userspace and the kernel, page faults, and context switches), as well as `taskstats` metrics (use these to obtain information about I/O wait time, CPU wait time, and the amount of data read and recorded, both with and without page cache). [#2482](#)
- The `ProfileEvents` counters are applied globally and for each query, as well as for each query execution thread, which allows you to profile resource consumption by query in detail. [#2482](#)
- Added the `system.query_thread_log` table, which contains information about each query execution thread. Added the `log_query_threads` setting. [#2482](#)
- The `system.metrics` and `system.events` tables now have built-in documentation. [#3016](#)
- Added the `arrayEnumerateDense` function. [Amos Bird](#)
- Added the `arrayCumSumNonNegative` and `arrayDifference` functions. [Aleksey Studnev](#)
- Added the `retention` aggregate function. [Sundy Li](#)
- Now you can add (merge) states of aggregate functions by using the plus operator, and multiply the states of aggregate functions by a nonnegative constant. [#3062](#) [#3034](#)
- Tables in the MergeTree family now have the virtual column `_partition_id`. [#3089](#)

## Experimental Features:

- Added the `LowCardinality(T)` data type. This data type automatically creates a local dictionary of values and allows data processing without unpacking the dictionary. [#2830](#)
- Added a cache of JIT-compiled functions and a counter for the number of uses before compiling. To JIT compile expressions, enable the `compile_expressions` setting. [#2990](#) [#3077](#)

## Improvements:

- Fixed the problem with unlimited accumulation of the replication log when there are abandoned replicas. Added an effective recovery mode for replicas with a long lag.
- Improved performance of `GROUP BY` with multiple aggregation fields when one of them is string and the others are fixed length.
- Improved performance when using `PREWHERE` and with implicit transfer of expressions in `PREWHERE`.
- Improved parsing performance for text formats (CSV, TSV). [Amos Bird](#) [#2980](#)
- Improved performance of reading strings and arrays in binary formats. [Amos Bird](#)
- Increased performance and reduced memory consumption for queries to `system.tables` and `system.columns` when there is a very large number of tables on a single server. [#2953](#)
- Fixed a performance problem in the case of a large stream of queries that result in an error (the `_dl_addr` function is visible in `perf top`, but the server isn't using much CPU). [#2938](#)
- Conditions are cast into the View (when `enable_optimize_predicate_expression` is enabled). [Winter Zhang](#)
- Improvements to the functionality for the `UUID` data type. [#3074](#) [#2985](#)
- The `UUID` data type is supported in The-Alchemist dictionaries. [#2822](#)
- The `visitParamExtractRaw` function works correctly with nested structures. [Winter Zhang](#)
- When the `input_format_skip_unknown_fields` setting is enabled, object fields in `JSONEachRow` format are skipped correctly. [BlahGeek](#)
- For a `CASE` expression with conditions, you can now omit `ELSE`, which is equivalent to `ELSE NULL`. [#2920](#)
- The operation timeout can now be configured when working with ZooKeeper. [urykhy](#)
- You can specify an offset for `LIMIT n, m` as `LIMIT n OFFSET m`. [#2840](#)
- You can use the `SELECT TOP n` syntax as an alternative for `LIMIT`. [#2840](#)
- Increased the size of the queue to write to system tables, so the `SystemLog` parameter `queue is full` error does not happen as often.
- The `windowFunnel` aggregate function now supports events that meet multiple conditions. [Amos Bird](#)
- Duplicate columns can be used in a `USING` clause for `JOIN`. [#3006](#)
- Pretty formats now have a limit on column alignment by width. Use the `output_format_pretty_max_column_pad_width` setting. If a value is wider, it will still be displayed in its entirety, but the other cells in the table will not be too wide. [#3003](#)
- The `odbc` table function now allows you to specify the database/schema name. [Amos Bird](#)
- Added the ability to use a username specified in the `clickhouse-client` config file. [Vladimir Kozbin](#)

- The `ZooKeeperExceptions` counter has been split into three counters: `ZooKeeperUserExceptions`, `ZooKeeperHardwareExceptions`, and `ZooKeeperOtherExceptions`.
- `ALTER DELETE` queries work for materialized views.
- Added randomization when running the cleanup thread periodically for `ReplicatedMergeTree` tables in order to avoid periodic load spikes when there are a very large number of `ReplicatedMergeTree` tables.
- Support for `ATTACH TABLE ... ON CLUSTER` queries. [#3025](#)

## Bug Fixes:

- Fixed an issue with `Dictionary` tables (throws the `Size of offsets does not match size of column or Unknown compression method` exception). This bug appeared in version 18.10.3. [#2913](#)
- Fixed a bug when merging `CollapsingMergeTree` tables if one of the data parts is empty (these parts are formed during merge or `ALTER DELETE` if all data was deleted), and the `vertical` algorithm was used for the merge. [#3049](#)
- Fixed a race condition during `DROP` or `TRUNCATE` for `Memory` tables with a simultaneous `SELECT`, which could lead to server crashes. This bug appeared in version 1.1.54388. [#3038](#)
- Fixed the possibility of data loss when inserting in `Replicated` tables if the `Session is expired` error is returned (data loss can be detected by the `ReplicatedDataLoss` metric). This error occurred in version 1.1.54378. [#2939](#) [#2949](#) [#2964](#)
- Fixed a segfault during `JOIN ... ON`. [#3000](#)
- Fixed the error searching column names when the `WHERE` expression consists entirely of a qualified column name, such as `WHERE table.column`. [#2994](#)
- Fixed the “Not found column” error that occurred when executing distributed queries if a single column consisting of an `IN` expression with a subquery is requested from a remote server. [#3087](#)
- Fixed the `Block structure mismatch in UNION stream: different number of columns` error that occurred for distributed queries if one of the shards is local and the other is not, and optimization of the move to `PREWHERE` is triggered. [#2226](#) [#3037](#) [#3055](#) [#3065](#) [#3073](#) [#3090](#) [#3093](#)
- Fixed the `pointInPolygon` function for certain cases of non-convex polygons. [#2910](#)
- Fixed the incorrect result when comparing `nan` with integers. [#3024](#)
- Fixed an error in the `zlib-ng` library that could lead to segfault in rare cases. [#2854](#)
- Fixed a memory leak when inserting into a table with `AggregateFunction` columns, if the state of the aggregate function is not simple (allocates memory separately), and if a single insertion request results in multiple small blocks. [#3084](#)
- Fixed a race condition when creating and deleting the same `Buffer` or `MergeTree` table simultaneously.
- Fixed the possibility of a segfault when comparing tuples made up of certain non-trivial types, such as tuples. [#2989](#)
- Fixed the possibility of a segfault when running certain `ON CLUSTER` queries. [Winter Zhang](#)
- Fixed an error in the `arrayDistinct` function for `Nullable` array elements. [#2845](#) [#2937](#)
- The `enable_optimize_predicate_expression` option now correctly supports cases with `SELECT *`. [Winter Zhang](#)
- Fixed the segfault when re-initializing the ZooKeeper session. [#2917](#)
- Fixed potential blocking when working with ZooKeeper.

- Fixed incorrect code for adding nested data structures in a `SummingMergeTree`.
- When allocating memory for states of aggregate functions, alignment is correctly taken into account, which makes it possible to use operations that require alignment when implementing states of aggregate functions. [chenxing-xc](#)

#### Security Fix:

- Safe use of ODBC data sources. Interaction with ODBC drivers uses a separate `clickhouse-odbc-bridge` process. Errors in third-party ODBC drivers no longer cause problems with server stability or vulnerabilities. [#2828](#) [#2879](#) [#2886](#) [#2893](#) [#2921](#)
- Fixed incorrect validation of the file path in the `catBoostPool` table function. [#2894](#)
- The contents of system tables (`tables`, `databases`, `parts`, `columns`, `parts_columns`, `merges`, `mutations`, `replicas`, and `replication_queue`) are filtered according to the user's configured access to databases (`allow_databases`). [Winter Zhang](#)

#### Backward Incompatible Changes:

- In queries with `JOIN`, the star character expands to a list of columns in all tables, in compliance with the SQL standard. You can restore the old behavior by setting `asterisk_left_columns_only` to 1 on the user configuration level.

#### Build Changes:

- Most integration tests can now be run by commit.
- Code style checks can also be run by commit.
- The `memcpy` implementation is chosen correctly when building on CentOS7/Fedora. [Etienne Champetier](#)
- When using clang to build, some warnings from `-Weverything` have been added, in addition to the regular `-Wall-Wextra -Werror`. [#2957](#)
- Debugging the build uses the `jemalloc` debug option.
- The interface of the library for interacting with ZooKeeper is declared abstract. [#2950](#)

## ClickHouse Release 18.10

### ClickHouse Release 18.10.3, 2018-08-13

#### New Features:

- HTTPS can be used for replication. [#2760](#)
- Added the functions `murmurHash2_64`, `murmurHash3_32`, `murmurHash3_64`, and `murmurHash3_128` in addition to the existing `murmurHash2_32`. [#2791](#)
- Support for Nullable types in the ClickHouse ODBC driver (ODBCDriver2 output format). [#2834](#)
- Support for `UUID` in the key columns.

#### Improvements:

- Clusters can be removed without restarting the server when they are deleted from the config files. [#2777](#)
- External dictionaries can be removed without restarting the server when they are removed from config files. [#2779](#)
- Added `SETTINGS` support for the `Kafka` table engine. [Alexander Marshalov](#)

- Improvements for the `UUID` data type (not yet complete). [#2618](#)
- Support for empty parts after merges in the `SummingMergeTree`, `CollapsingMergeTree` and `VersionedCollapsingMergeTree` engines. [#2815](#)
- Old records of completed mutations are deleted (`ALTER DELETE`). [#2784](#)
- Added the `system.merge_tree_settings` table. [Kirill Shvakov](#)
- The `system.tables` table now has dependency columns: `dependencies_database` and `dependencies_table`. [Winter Zhang](#)
- Added the `max_partition_size_to_drop` config option. [#2782](#)
- Added the `output_format_json_escape_forward_slashes` option. [Alexander Bocharov](#)
- Added the `max_fetch_partition_retries_count` setting. [#2831](#)
- Added the `prefer_localhost_replica` setting for disabling the preference for a local replica and going to a local replica without inter-process interaction. [#2832](#)
- The `quantileExact` aggregate function returns `nan` in the case of aggregation on an empty `Float32` or `Float64` set. [Sundy Li](#)

## Bug Fixes:

- Removed unnecessary escaping of the connection string parameters for ODBC, which made it impossible to establish a connection. This error occurred in version 18.6.0.
- Fixed the logic for processing `REPLACE PARTITION` commands in the replication queue. If there are two `REPLACE` commands for the same partition, the incorrect logic could cause one of them to remain in the replication queue and not be executed. [#2814](#)
- Fixed a merge bug when all data parts were empty (parts that were formed from a merge or from `ALTER DELETE` if all data was deleted). This bug appeared in version 18.1.0. [#2930](#)
- Fixed an error for concurrent `Set` or `Join`. [Amos Bird](#)
- Fixed the `Block structure mismatch in UNION stream: different number of columns` error that occurred for `UNION ALL` queries inside a sub-query if one of the `SELECT` queries contains duplicate column names. [Winter Zhang](#)
- Fixed a memory leak if an exception occurred when connecting to a MySQL server.
- Fixed incorrect clickhouse-client response code in case of a query error.
- Fixed incorrect behavior of materialized views containing `DISTINCT`. [#2795](#)

## Backward Incompatible Changes

- Removed support for `CHECK TABLE` queries for Distributed tables.

## Build Changes:

- The allocator has been replaced: `jemalloc` is now used instead of `tcmalloc`. In some scenarios, this increases speed up to 20%. However, there are queries that have slowed by up to 20%. Memory consumption has been reduced by approximately 10% in some scenarios, with improved stability. With highly competitive loads, CPU usage in userspace and in system shows just a slight increase. [#2773](#)
- Use of `libressl` from a submodule. [#1983 #2807](#)
- Use of `unixodbc` from a submodule. [#2789](#)

- Use of mariadb-connector-c from a submodule. [#2785](#)
- Added functional test files to the repository that depend on the availability of test data (for the time being, without the test data itself).

## ClickHouse Release 18.6

### ClickHouse Release 18.6.0, 2018-08-02

#### New Features:

- Added support for ON expressions for the JOIN ON syntax:

```
JOIN ON Expr([table.]column ...) = Expr([table.]column, ...) [AND Expr([table.]column, ...) = Expr([table.]column, ...) ...]
```

The expression must be a chain of equalities joined by the AND operator. Each side of the equality can be an arbitrary expression over the columns of one of the tables. The use of fully qualified column names is supported (table.name, database.table.name, table\_alias.name, subquery\_alias.name) for the right table. [#2742](#)

- HTTPS can be enabled for replication. [#2760](#)

#### Improvements:

- The server passes the patch component of its version to the client. Data about the patch version component is in `system.processes` and `query_log`. [#2646](#)

## ClickHouse Release 18.5

### ClickHouse Release 18.5.1, 2018-07-31

#### New Features:

- Added the hash function `murmurHash2_32` [#2756](#).

#### Improvements:

- Now you can use the `from_env` [#2741](#) attribute to set values in config files from environment variables.
- Added case-insensitive versions of the `coalesce`, `ifNull`, and `nullIf` functions [#2752](#).

#### Bug Fixes:

- Fixed a possible bug when starting a replica [#2759](#).

## ClickHouse Release 18.4

### ClickHouse Release 18.4.0, 2018-07-28

#### New Features:

- Added system tables: `formats`, `data_type_families`, `aggregate_function_combinators`, `table_functions`, `table_engines`, `collations` [#2721](#).
- Added the ability to use a table function instead of a table as an argument of a `remote` or `cluster` table function [#2708](#).
- Support for HTTP Basic authentication in the replication protocol [#2727](#).
- The `has` function now allows searching for a numeric value in an array of `Enum` values [Maxim Khrisanfov](#).
- Support for adding arbitrary message separators when reading from Kafka [Amos Bird](#).

#### Improvements:

- The `ALTER TABLE t DELETE WHERE` query does not rewrite data parts that were not affected by the `WHERE` condition [#2694](#).
- The `use_minimalistic_checksums_in_zookeeper` option for `ReplicatedMergeTree` tables is enabled by default. This setting was added in version 1.1.54378, 2018-04-16. Versions that are older than 1.1.54378 can no longer be installed.
- Support for running `KILL` and `OPTIMIZE` queries that specify `ON CLUSTER` [Winter Zhang](#).

#### Bug Fixes:

- Fixed the error `Column ... is not under an aggregate function and not in GROUP BY` for aggregation with an `IN` expression. This bug appeared in version 18.1.0. ([bbdd780b](#))
- Fixed a bug in the `windowFunnel` aggregate function [Winter Zhang](#).
- Fixed a bug in the `anyHeavy` aggregate function ([a2101df2](#))
- Fixed server crash when using the `countArray()` aggregate function.

#### Backward Incompatible Changes:

- Parameters for `Kafka` engine was changed from `Kafka(kafka_broker_list, kafka_topic_list, kafka_group_name, kafka_format[, kafka_schema, kafka_num_consumers])` to `Kafka(kafka_broker_list, kafka_topic_list, kafka_group_name, kafka_format[, kafka_row_delimiter, kafka_schema, kafka_num_consumers])`. If your tables use `kafka_schema` or `kafka_num_consumers` parameters, you have to manually edit the metadata files `path/metadata/database/table.sql` and add `kafka_row_delimiter` parameter with `" "` value.

## ClickHouse Release 18.1

### ClickHouse Release 18.1.0, 2018-07-23

#### New Features:

- Support for the `ALTER TABLE t DELETE WHERE` query for non-replicated `MergeTree` tables ([#2634](#)).
- Support for arbitrary types for the `uniq*` family of aggregate functions ([#2010](#)).
- Support for arbitrary types in comparison operators ([#2026](#)).
- The `users.xml` file allows setting a subnet mask in the format `10.0.0.1/255.255.255.0`. This is necessary for using masks for IPv6 networks with zeros in the middle ([#2637](#)).
- Added the `arrayDistinct` function ([#2670](#)).
- The `SummingMergeTree` engine can now work with `AggregateFunction` type columns ([Constantin S. Pan](#)).

#### Improvements:

- Changed the numbering scheme for release versions. Now the first part contains the year of release (A.D., Moscow timezone, minus 2000), the second part contains the number for major changes (increases for most releases), and the third part is the patch version. Releases are still backward compatible, unless otherwise stated in the changelog.
- Faster conversions of floating-point numbers to a string ([Amos Bird](#)).
- If some rows were skipped during an insert due to parsing errors (this is possible with the `input_allow_errors_num` and `input_allow_errors_ratio` settings enabled), the number of skipped rows is now written to the server log ([Leonardo Cecchi](#)).

#### Bug Fixes:

- Fixed the TRUNCATE command for temporary tables ([Amos Bird](#)).
- Fixed a rare deadlock in the ZooKeeper client library that occurred when there was a network error while reading the response ([c315200](#)).
- Fixed an error during a CAST to Nullable types ([#1322](#)).
- Fixed the incorrect result of the `maxIntersection()` function when the boundaries of intervals coincided ([Michael Furmur](#)).
- Fixed incorrect transformation of the OR expression chain in a function argument ([chenxing-xc](#)).
- Fixed performance degradation for queries containing `IN (subquery)` expressions inside another subquery ([#2571](#)).
- Fixed incompatibility between servers with different versions in distributed queries that use a `CAST` function that isn't in uppercase letters ([fe8c4d6](#)).
- Added missing quoting of identifiers for queries to an external DBMS ([#2635](#)).

#### Backward Incompatible Changes:

- Converting a string containing the number zero to `DateTime` does not work. Example: `SELECT toDateTime('0')`. This is also the reason that `DateTime DEFAULT '0'` does not work in tables, as well as `<null_value>0</null_value>` in dictionaries. Solution: replace 0 with `0000-00-00 00:00:00`.

## ClickHouse Release 1.1

### ClickHouse Release 1.1.54394, 2018-07-12

#### New Features:

- Added the `histogram` aggregate function ([Mikhail Surin](#)).
- Now `OPTIMIZE TABLE ... FINAL` can be used without specifying partitions for `ReplicatedMergeTree` ([Amos Bird](#)).

#### Bug Fixes:

- Fixed a problem with a very small timeout for sockets (one second) for reading and writing when sending and downloading replicated data, which made it impossible to download larger parts if there is a load on the network or disk (it resulted in cyclical attempts to download parts). This error occurred in version 1.1.54388.
- Fixed issues when using chroot in ZooKeeper if you inserted duplicate data blocks in the table.
- The `has` function now works correctly for an array with Nullable elements ([#2115](#)).
- The `system.tables` table now works correctly when used in distributed queries. The `metadata_modification_time` and `engine_full` columns are now non-virtual. Fixed an error that occurred if only these columns were queried from the table.
- Fixed how an empty `TinyLog` table works after inserting an empty data block ([#2563](#)).
- The `system.zookeeper` table works if the value of the node in ZooKeeper is `NULL`.

### ClickHouse Release 1.1.54390, 2018-07-06

#### New Features:

- Queries can be sent in multipart/form-data format (in the `query` field), which is useful if external data is also sent for query processing ([Olga Hvostikova](#)).

- Added the ability to enable or disable processing single or double quotes when reading data in CSV format. You can configure this in the `format_csv_allow_single_quotes` and `format_csv_allow_double_quotes` settings ([Amos Bird](#)).
- Now `OPTIMIZE TABLE ... FINAL` can be used without specifying the partition for non-replicated variants of `MergeTree` ([Amos Bird](#)).

## Improvements:

- Improved performance, reduced memory consumption, and correct memory consumption tracking with use of the IN operator when a table index could be used ([#2584](#)).
- Removed redundant checking of checksums when adding a data part. This is important when there are a large number of replicas, because in these cases the total number of checks was equal to  $N^2$ .
- Added support for `Array(Tuple(...))` arguments for the `arrayEnumerateUniq` function ([#2573](#)).
- Added `Nullable` support for the `runningDifference` function ([#2594](#)).
- Improved query analysis performance when there is a very large number of expressions ([#2572](#)).
- Faster selection of data parts for merging in `ReplicatedMergeTree` tables. Faster recovery of the ZooKeeper session ([#2597](#)).
- The `format_version.txt` file for `MergeTree` tables is re-created if it is missing, which makes sense if ClickHouse is launched after copying the directory structure without files ([Ciprian Hacman](#)).

## Bug Fixes:

- Fixed a bug when working with ZooKeeper that could make it impossible to recover the session and readonly states of tables before restarting the server.
- Fixed a bug when working with ZooKeeper that could result in old nodes not being deleted if the session is interrupted.
- Fixed an error in the `quantileTDigest` function for Float arguments (this bug was introduced in version 1.1.54388) ([Mikhail Surin](#)).
- Fixed a bug in the index for MergeTree tables if the primary key column is located inside the function for converting types between signed and unsigned integers of the same size ([#2603](#)).
- Fixed segfault if `macros` are used but they aren't in the config file ([#2570](#)).
- Fixed switching to the default database when reconnecting the client ([#2583](#)).
- Fixed a bug that occurred when the `use_index_for_in_with_subqueries` setting was disabled.

## Security Fix:

- Sending files is no longer possible when connected to MySQL (`LOAD DATA LOCAL INFILE`).

## ClickHouse Release 1.1.54388, 2018-06-28

### New Features:

- Support for the `ALTER TABLE t DELETE WHERE` query for replicated tables. Added the `system.mutations` table to track progress of this type of queries.
- Support for the `ALTER TABLE t [REPLACE|ATTACH] PARTITION` query for \*MergeTree tables.
- Support for the `TRUNCATE TABLE` query ([Winter Zhang](#))

- Several new `SYSTEM` queries for replicated tables (`RESTART REPLICAS`, `SYNC REPLICA`, [`STOP|START`], [`MERGES|FETCHES|SENDS REPLICATED|REPLICATION QUEUES`]).
- Added the ability to write to a table with the MySQL engine and the corresponding table function ([sundy-li](#)).
- Added the `url()` table function and the `URL` table engine ([Alexander Sapin](#)).
- Added the `windowFunnel` aggregate function ([sundy-li](#)).
- New `startsWith` and `endsWith` functions for strings ([Vadim Plakhtinsky](#)).
- The `numbers()` table function now allows you to specify the offset ([Winter Zhang](#)).
- The password to `clickhouse-client` can be entered interactively.
- Server logs can now be sent to syslog ([Alexander Krasheninnikov](#)).
- Support for logging in dictionaries with a shared library source ([Alexander Sapin](#)).
- Support for custom CSV delimiters ([Ivan Zhukov](#))
- Added the `date_time_input_format` setting. If you switch this setting to 'best\_effort', `DateTime` values will be read in a wide range of formats.
- Added the `clickhouse-obfuscator` utility for data obfuscation. Usage example: publishing data used in performance tests.

## Experimental Features:

- Added the ability to calculate `and` arguments only where they are needed ([Anastasia Tsarkova](#))
- JIT compilation to native code is now available for some expressions ([pyos](#)).

## Bug Fixes:

- Duplicates no longer appear for a query with `DISTINCT` and `ORDER BY`.
- Queries with `ARRAY JOIN` and `arrayFilter` no longer return an incorrect result.
- Fixed an error when reading an array column from a Nested structure ([#2066](#)).
- Fixed an error when analyzing queries with a `HAVING` clause like `HAVING tuple IN (...)`.
- Fixed an error when analyzing queries with recursive aliases.
- Fixed an error when reading from `ReplacingMergeTree` with a condition in `PREWHERE` that filters all rows ([#2525](#)).
- User profile settings were not applied when using sessions in the HTTP interface.
- Fixed how settings are applied from the command line parameters in `clickhouse-local`.
- The ZooKeeper client library now uses the session timeout received from the server.
- Fixed a bug in the ZooKeeper client library when the client waited for the server response longer than the timeout.
- Fixed pruning of parts for queries with conditions on partition key columns ([#2342](#)).
- Merges are now possible after `CLEAR COLUMN IN PARTITION` ([#2315](#)).
- Type mapping in the ODBC table function has been fixed ([sundy-li](#)).

- Type comparisons have been fixed for `DateTime` with and without the time zone ([Alexander Bocharov](#)).
- Fixed syntactic parsing and formatting of the `CAST` operator.
- Fixed insertion into a materialized view for the Distributed table engine ([Babacar Diassé](#)).
- Fixed a race condition when writing data from the `Kafka` engine to materialized views ([Yangkuan Liu](#)).
- Fixed SSRF in the `remote()` table function.
- Fixed exit behavior of `clickhouse-client` in multiline mode ([#2510](#)).

## Improvements:

- Background tasks in replicated tables are now performed in a thread pool instead of in separate threads ([Silviu Caragea](#)).
- Improved LZ4 compression performance.
- Faster analysis for queries with a large number of JOINs and sub-queries.
- The DNS cache is now updated automatically when there are too many network errors.
- Table inserts no longer occur if the insert into one of the materialized views is not possible because it has too many parts.
- Corrected the discrepancy in the event counters `Query`, `SelectQuery`, and `InsertQuery`.
- Expressions like `tuple IN (SELECT tuple)` are allowed if the tuple types match.
- A server with replicated tables can start even if you haven't configured ZooKeeper.
- When calculating the number of available CPU cores, limits on cgroups are now taken into account ([Atri Sharma](#)).
- Added chown for config directories in the systemd config file ([Mikhail Shiryaev](#)).

## Build Changes:

- The `gcc8` compiler can be used for builds.
- Added the ability to build `llvm` from submodule.
- The version of the `librdkafka` library has been updated to v0.11.4.
- Added the ability to use the system `libcpuid` library. The library version has been updated to 0.4.0.
- Fixed the build using the `vectorclass` library ([Babacar Diassé](#)).
- Cmake now generates files for `ninja` by default (like when using `-G Ninja`).
- Added the ability to use the `libtinfo` library instead of `libtermcap` ([Georgy Kondratiev](#)).
- Fixed a header file conflict in Fedora Rawhide ([#2520](#)).

## Backward Incompatible Changes:

- Removed escaping in `Vertical` and `Pretty*` formats and deleted the `VerticalRaw` format.
- If servers with version 1.1.54388 (or newer) and servers with an older version are used simultaneously in a distributed query and the query has the `cast(x, 'Type')` expression without the `AS` keyword and does not have the word `cast` in uppercase, an exception will be thrown with a message like `Not found column cast(0, 'UInt8') in block`. Solution: Update the server on the entire cluster.

## ClickHouse Release 1.1.54385, 2018-06-01

### Bug Fixes:

- Fixed an error that in some cases caused ZooKeeper operations to block.

## ClickHouse Release 1.1.54383, 2018-05-22

### Bug Fixes:

- Fixed a slowdown of replication queue if a table has many replicas.

## ClickHouse Release 1.1.54381, 2018-05-14

### Bug Fixes:

- Fixed a nodes leak in ZooKeeper when ClickHouse loses connection to ZooKeeper server.

## ClickHouse Release 1.1.54380, 2018-04-21

### New Features:

- Added the table function `file(path, format, structure)`. An example reading bytes from `/dev/urandom`:  
`In -s /dev/urandom /var/lib/clickhouse/user_files/random``clickhouse-client -q "SELECT * FROM file('random', 'RowBinary', 'd UInt8') LIMIT 10".`

### Improvements:

- Subqueries can be wrapped in `()` brackets to enhance query readability. For example: `(SELECT 1) UNION ALL (SELECT 1)`.
- Simple `SELECT` queries from the `system.processes` table are not included in the `max_concurrent_queries` limit.

### Bug Fixes:

- Fixed incorrect behavior of the `IN` operator when select from `MATERIALIZED VIEW`.
- Fixed incorrect filtering by partition index in expressions like `partition_key_column IN (...)`.
- Fixed inability to execute `OPTIMIZE` query on non-leader replica if `RENAME` was performed on the table.
- Fixed the authorization error when executing `OPTIMIZE` or `ALTER` queries on a non-leader replica.
- Fixed freezing of `KILL QUERY`.
- Fixed an error in ZooKeeper client library which led to loss of watches, freezing of distributed DDL queue, and slowdowns in the replication queue if a non-empty `chroot` prefix is used in the ZooKeeper configuration.

### Backward Incompatible Changes:

- Removed support for expressions like `(a, b) IN (SELECT (a, b))` (you can use the equivalent expression `(a, b) IN (SELECT a, b)`). In previous releases, these expressions led to undetermined `WHERE` filtering or caused errors.

## ClickHouse Release 1.1.54378, 2018-04-16

### New Features:

- Logging level can be changed without restarting the server.
- Added the `SHOW CREATE DATABASE` query.
- The `query_id` can be passed to `clickhouse-client` (`elBroom`).

- New setting: `max_network_bandwidth_for_all_users`.
- Added support for `ALTER TABLE ... PARTITION ...` for `MATERIALIZED VIEW`.
- Added information about the size of data parts in uncompressed form in the system table.
- Server-to-server encryption support for distributed tables (`<secure>1</secure>` in the replica config in `<remote_servers>`).
- Configuration of the table level for the `ReplicatedMergeTree` family in order to minimize the amount of data stored in Zookeeper: : `use_minimalistic_checksums_in_zookeeper = 1`
- Configuration of the `clickhouse-client` prompt. By default, server names are now output to the prompt. The server's display name can be changed. It's also sent in the `X-ClickHouse-Display-Name` HTTP header (Kirill Shvakov).
- Multiple comma-separated `topics` can be specified for the `Kafka` engine (Tobias Adamson)
- When a query is stopped by `KILL QUERY` or `replace_running_query`, the client receives the `Query was canceled` exception instead of an incomplete result.

### Improvements:

- `ALTER TABLE ... DROP/DETACH PARTITION` queries are run at the front of the replication queue.
- `SELECT ... FINAL` and `OPTIMIZE ... FINAL` can be used even when the table has a single data part.
- A `query_log` table is recreated on the fly if it was deleted manually (Kirill Shvakov).
- The `lengthUTF8` function runs faster (zhang2014).
- Improved performance of synchronous inserts in `Distributed` tables (`insert_distributed_sync = 1`) when there is a very large number of shards.
- The server accepts the `send_timeout` and `receive_timeout` settings from the client and applies them when connecting to the client (they are applied in reverse order: the server socket's `send_timeout` is set to the `receive_timeout` value received from the client, and vice versa).
- More robust crash recovery for asynchronous insertion into `Distributed` tables.
- The return type of the `countEqual` function changed from `UInt32` to `UInt64` (谢磊).

### Bug Fixes:

- Fixed an error with `IN` when the left side of the expression is `Nullable`.
- Correct results are now returned when using tuples with `IN` when some of the tuple components are in the table index.
- The `max_execution_time` limit now works correctly with distributed queries.
- Fixed errors when calculating the size of composite columns in the `system.columns` table.
- Fixed an error when creating a temporary table `CREATE TEMPORARY TABLE IF NOT EXISTS`.
- Fixed errors in `StorageKafka` (#2075)
- Fixed server crashes from invalid arguments of certain aggregate functions.
- Fixed the error that prevented the `DETACH DATABASE` query from stopping background tasks for `ReplicatedMergeTree` tables.

- Too many parts state is less likely to happen when inserting into aggregated materialized views (#2084).
- Corrected recursive handling of substitutions in the config if a substitution must be followed by another substitution on the same level.
- Corrected the syntax in the metadata file when creating a `VIEW` that uses a query with `UNION ALL`.
- SummingMergeTree now works correctly for summation of nested data structures with a composite key.
- Fixed the possibility of a race condition when choosing the leader for ReplicatedMergeTree tables.

#### Build Changes:

- The build supports `ninja` instead of `make` and uses `ninja` by default for building releases.
- Renamed packages: `clickhouse-server-base` in `clickhouse-common-static`; `clickhouse-server-common` in `clickhouse-server`; `clickhouse-common-dbg` in `clickhouse-common-static-dbg`. To install, use `clickhouse-server` `clickhouse-client`. Packages with the old names will still load in the repositories for backward compatibility.

#### Backward Incompatible Changes:

- Removed the special interpretation of an IN expression if an array is specified on the left side. Previously, the expression `arr IN (set)` was interpreted as “at least one `arr` element belongs to the `set`”. To get the same behavior in the new version, write `arrayExists(x -> x IN (set), arr)`.
- Disabled the incorrect use of the socket option `SO_REUSEPORT`, which was incorrectly enabled by default in the Poco library. Note that on Linux there is no longer any reason to simultaneously specify the addresses `::` and `0.0.0.0` for listen – use just `::`, which allows listening to the connection both over IPv4 and IPv6 (with the default kernel config settings). You can also revert to the behavior from previous versions by specifying `<listen_reuse_port>1</listen_reuse_port>` in the config.

## ClickHouse Release 1.1.54370, 2018-03-16

#### New Features:

- Added the `system.macros` table and auto updating of macros when the config file is changed.
- Added the `SYSTEM RELOAD CONFIG` query.
- Added the `maxIntersections(left_col, right_col)` aggregate function, which returns the maximum number of simultaneously intersecting intervals `[left; right]`. The `maxIntersectionsPosition(left, right)` function returns the beginning of the “maximum” interval. ([Michael Furmur](#)).

#### Improvements:

- When inserting data in a Replicated table, fewer requests are made to ZooKeeper (and most of the user-level errors have disappeared from the ZooKeeper log).
- Added the ability to create aliases for data sets. Example: `WITH (1, 2, 3) AS set SELECT number IN set FROM system.numbers LIMIT 10`.

#### Bug Fixes:

- Fixed the `Illegal PREWHERE` error when reading from Merge tables for Distributedtables.
- Added fixes that allow you to start clickhouse-server in IPv4-only Docker containers.
- Fixed a race condition when reading from system `system.parts_columns` tables.
- Removed double buffering during a synchronous insert to a `Distributed` table, which could have caused the connection to timeout.

- Fixed a bug that caused excessively long waits for an unavailable replica before beginning a `SELECT` query.
- Fixed incorrect dates in the `system.parts` table.
- Fixed a bug that made it impossible to insert data in a `Replicated` table if `chroot` was non-empty in the configuration of the `ZooKeeper` cluster.
- Fixed the vertical merging algorithm for an empty `ORDER BY` table.
- Restored the ability to use dictionaries in queries to remote tables, even if these dictionaries are not present on the requestor server. This functionality was lost in release 1.1.54362.
- Restored the behavior for queries like `SELECT * FROM remote('server2', default.table) WHERE col IN (SELECT col2 FROM default.table)` when the right side of the `IN` should use a remote `default.table` instead of a local one. This behavior was broken in version 1.1.54358.
- Removed extraneous error-level logging of `Not found column ... in block`

## ClickHouse Release 1.1.54362, 2018-03-11

### New Features:

- Aggregation without `GROUP BY` for an empty set (such as `SELECT count(*) FROM table WHERE 0`) now returns a result with one row with null values for aggregate functions, in compliance with the SQL standard. To restore the old behavior (return an empty result), set `empty_result_for_aggregation_by_empty_set` to 1.
- Added type conversion for `UNION ALL`. Different alias names are allowed in `SELECT` positions in `UNION ALL`, in compliance with the SQL standard.
- Arbitrary expressions are supported in `LIMIT BY` clauses. Previously, it was only possible to use columns resulting from `SELECT`.
- An index of `MergeTree` tables is used when `IN` is applied to a tuple of expressions from the columns of the primary key. Example: `WHERE (UserID, EventDate) IN ((123, '2000-01-01'), ...)` (Anastasiya Tsarkova).
- Added the `clickhouse-copier` tool for copying between clusters and resharding data (beta).
- Added consistent hashing functions: `yandexConsistentHash`, `jumpConsistentHash`, `sumburConsistentHash`. They can be used as a sharding key in order to reduce the amount of network traffic during subsequent reshardings.
- Added functions: `arrayAny`, `arrayAll`, `hasAny`, `hasAll`, `arrayIntersect`, `arrayResize`.
- Added the `arrayCumSum` function (Javi Santana).
- Added the `parseDateTimeBestEffort`, `parseDateTimeBestEffortOrZero`, and `parseDateTimeBestEffortOrNull` functions to read the `DateTime` from a string containing text in a wide variety of possible formats.
- Data can be partially reloaded from external dictionaries during updating (load just the records in which the value of the specified field greater than in the previous download) (Arsen Hakobyan).
- Added the `cluster` table function. Example: `cluster(cluster_name, db, table)`. The `remote` table function can accept the cluster name as the first argument, if it is specified as an identifier.
- The `remote` and `cluster` table functions can be used in `INSERT` queries.
- Added the `create_table_query` and `engine_full` virtual columns to the `system.tablestable` . The `metadata_modification_time` column is virtual.

- Added the `data_path` and `metadata_path` columns to `system.tables` and `system.databases` tables, and added the `path` column to the `system.parts` and `system.parts_columns` tables.
- Added additional information about merges in the `system.part_log` table.
- An arbitrary partitioning key can be used for the `system.query_log` table (Kirill Shvakov).
- The `SHOW TABLES` query now also shows temporary tables. Added temporary tables and the `is_temporary` column to `system.tables` (zhang2014).
- Added `DROP TEMPORARY TABLE` and `EXISTS TEMPORARY TABLE` queries (zhang2014).
- Support for `SHOW CREATE TABLE` for temporary tables (zhang2014).
- Added the `system_profile` configuration parameter for the settings used by internal processes.
- Support for loading `object_id` as an attribute in `MongoDB` dictionaries (Pavel Litvinenko).
- Reading `null` as the default value when loading data for an external dictionary with the `MongoDB` source (Pavel Litvinenko).
- Reading `DateTime` values in the `Values` format from a Unix timestamp without single quotes.
- Failover is supported in `remote` table functions for cases when some of the replicas are missing the requested table.
- Configuration settings can be overridden in the command line when you run `clickhouse-server`. Example: `clickhouse-server -- --logger.level=information`.
- Implemented the `empty` function from a `FixedString` argument: the function returns 1 if the string consists entirely of null bytes (zhang2014).
- Added the `listen_try` configuration parameter for listening to at least one of the listen addresses without quitting, if some of the addresses can't be listened to (useful for systems with disabled support for IPv4 or IPv6).
- Added the `VersionedCollapsingMergeTree` table engine.
- Support for rows and arbitrary numeric types for the `library` dictionary source.
- `MergeTree` tables can be used without a primary key (you need to specify `ORDER BY tuple()`).
- A `Nullable` type can be `CAST` to a non-`Nullable` type if the argument is not `NULL`.
- `RENAME TABLE` can be performed for `VIEW`.
- Added the `throwIf` function.
- Added the `odbc_default_field_size` option, which allows you to extend the maximum size of the value loaded from an ODBC source (by default, it is 1024).
- The `system.processes` table and `SHOW PROCESSLIST` now have the `is_cancelled` and `peak_memory_usage` columns.

## Improvements:

- Limits and quotas on the result are no longer applied to intermediate data for `INSERT SELECT` queries or for `SELECT` subqueries.
- Fewer false triggers of `force_restore_data` when checking the status of Replicated tables when the server starts.
- Added the `allow_distributed_ddl` option.

- Nondeterministic functions are not allowed in expressions for `MergeTree` table keys.
- Files with substitutions from `config.d` directories are loaded in alphabetical order.
- Improved performance of the `arrayElement` function in the case of a constant multidimensional array with an empty array as one of the elements. Example: `[[1], []][x]`.
- The server starts faster now when using configuration files with very large substitutions (for instance, very large lists of IP networks).
- When running a query, table valued functions run once. Previously, `remote` and `mysql` table valued functions performed the same query twice to retrieve the table structure from a remote server.
- The `MkDocs` documentation generator is used.
- When you try to delete a table column that `DEFAULT/MATERIALIZED` expressions of other columns depend on, an exception is thrown (zhang2014).
- Added the ability to parse an empty line in text formats as the number 0 for `Float` data types. This feature was previously available but was lost in release 1.1.54342.
- Enum values can be used in `min`, `max`, `sum` and some other functions. In these cases, it uses the corresponding numeric values. This feature was previously available but was lost in the release 1.1.54337.
- Added `max_expanded_ast_elements` to restrict the size of the AST after recursively expanding aliases.

## Bug Fixes:

- Fixed cases when unnecessary columns were removed from subqueries in error, or not removed from subqueries containing `UNION ALL`.
- Fixed a bug in merges for `ReplacingMergeTree` tables.
- Fixed synchronous insertions in `Distributed` tables (`insert_distributed_sync = 1`).
- Fixed segfault for certain uses of `FULL` and `RIGHT JOIN` with duplicate columns in subqueries.
- Fixed segfault for certain uses of `replace_running_query` and `KILL QUERY`.
- Fixed the order of the `source` and `last_exception` columns in the `system.dictionaries` table.
- Fixed a bug when the `DROP DATABASE` query did not delete the file with metadata.
- Fixed the `DROP DATABASE` query for `Dictionary` databases.
- Fixed the low precision of `uniqHLL12` and `uniqCombined` functions for cardinalities greater than 100 million items (Alex Bocharov).
- Fixed the calculation of implicit default values when necessary to simultaneously calculate default explicit expressions in `INSERT` queries (zhang2014).
- Fixed a rare case when a query to a `MergeTree` table couldn't finish (chenxing-xc).
- Fixed a crash that occurred when running a `CHECK` query for `Distributed` tables if all shards are local (chenxing.xc).
- Fixed a slight performance regression with functions that use regular expressions.
- Fixed a performance regression when creating multidimensional arrays from complex expressions.
- Fixed a bug that could cause an extra `FORMAT` section to appear in an `.sql` file with metadata.

- Fixed a bug that caused the `max_table_size_to_drop` limit to apply when trying to delete a `MATERIALIZED VIEW` looking at an explicitly specified table.
- Fixed incompatibility with old clients (old clients were sometimes sent data with the `DateTime('timezone')` type, which they do not understand).
- Fixed a bug when reading `Nested` column elements of structures that were added using `ALTER` but that are empty for the old partitions, when the conditions for these columns moved to `PREWHERE`.
- Fixed a bug when filtering tables by virtual `_table` columns in queries to `Merge` tables.
- Fixed a bug when using `ALIAS` columns in `Distributed` tables.
- Fixed a bug that made dynamic compilation impossible for queries with aggregate functions from the `quantile` family.
- Fixed a race condition in the query execution pipeline that occurred in very rare cases when using `Merge` tables with a large number of tables, and when using `GLOBAL` subqueries.
- Fixed a crash when passing arrays of different sizes to an `arrayReduce` function when using aggregate functions from multiple arguments.
- Prohibited the use of queries with `UNION ALL` in a `MATERIALIZED VIEW`.
- Fixed an error during initialization of the `part_log` system table when the server starts (by default, `part_log` is disabled).

#### Backward Incompatible Changes:

- Removed the `distributed_ddl_allow_replicated_alter` option. This behavior is enabled by default.
- Removed the `strict_insert_defaults` setting. If you were using this functionality, write to `clickhouse-feedback@yandex-team.com`.
- Removed the `UnsortedMergeTree` engine.

## ClickHouse Release 1.1.54343, 2018-02-05

- Added macros support for defining cluster names in distributed DDL queries and constructors of `Distributed` tables: `CREATE TABLE distr ON CLUSTER '{cluster}' (...) ENGINE = Distributed('{cluster}', 'db', 'table')`
- Now queries like `SELECT ... FROM table WHERE expr IN (subquery)` are processed using the `table` index.
- Improved processing of duplicates when inserting to Replicated tables, so they no longer slow down execution of the replication queue.

## ClickHouse Release 1.1.54342, 2018-01-22

This release contains bug fixes for the previous release 1.1.54337:

- Fixed a regression in 1.1.54337: if the default user has readonly access, then the server refuses to start up with the message `Cannot create database in readonly mode`.
- Fixed a regression in 1.1.54337: on systems with `systemd`, logs are always written to `syslog` regardless of the configuration; the `watchdog` script still uses `init.d`.
- Fixed a regression in 1.1.54337: wrong default configuration in the Docker image.
- Fixed nondeterministic behavior of `GraphiteMergeTree` (you can see it in log messages `Data after merge is not byte-identical to the data on another replicas`).

- Fixed a bug that may lead to inconsistent merges after OPTIMIZE query to Replicated tables (you may see it in log messages `Part ... intersects the previous part`).
- Buffer tables now work correctly when MATERIALIZED columns are present in the destination table (by zhang2014).
- Fixed a bug in implementation of NULL.

## ClickHouse Release 1.1.54337, 2018-01-18

### New Features:

- Added support for storage of multi-dimensional arrays and tuples (`Tuple` data type) in tables.
- Support for table functions for `DESCRIBE` and `INSERT` queries. Added support for subqueries in `DESCRIBE`. Examples: `DESC TABLE remote('host', default.hits); DESC TABLE (SELECT 1); INSERT INTO TABLE FUNCTION remote('host', default.hits)`. Support for `INSERT INTO TABLE` in addition to `INSERT INTO`.
- Improved support for time zones. The `DateTime` data type can be annotated with the timezone that is used for parsing and formatting in text formats. Example: `DateTime('Europe/Moscow')`. When timezones are specified in functions for `DateTime` arguments, the return type will track the timezone, and the value will be displayed as expected.
- Added the functions `toTimeZone`, `timeDiff`, `toQuarter`, `toRelativeQuarterNum`. The `toRelativeHour/Minute/Second` functions can take a value of type `Date` as an argument. The `now` function name is case-sensitive.
- Added the `toStartOfFifteenMinutes` function (Kirill Shvakov).
- Added the `clickhouse format` tool for formatting queries.
- Added the `format_schema_path` configuration parameter (Marek Vavruša). It is used for specifying a schema in `Cap'n Proto` format. Schema files can be located only in the specified directory.
- Added support for config substitutions (`incl` and `conf.d`) for configuration of external dictionaries and models (Pavel Yakunin).
- Added a column with documentation for the `system.settings` table (Kirill Shvakov).
- Added the `system.parts_columns` table with information about column sizes in each data part of `MergeTree` tables.
- Added the `system.models` table with information about loaded `CatBoost` machine learning models.
- Added the `mysql` and `odbc` table function and corresponding `MySQL` and `ODBC` table engines for accessing remote databases. This functionality is in the beta stage.
- Added the possibility to pass an argument of type `AggregateFunction` for the `groupArray` aggregate function (so you can create an array of states of some aggregate function).
- Removed restrictions on various combinations of aggregate function combinator. For example, you can use `avgForEachIf` as well as `avgIfForEach` aggregate functions, which have different behaviors.
- The `-ForEach` aggregate function combinator is extended for the case of aggregate functions of multiple arguments.
- Added support for aggregate functions of `Nullable` arguments even for cases when the function returns a non-`Nullable` result (added with the contribution of Silviu Caragea). Example: `groupArray`, `groupUniqArray`, `topK`.
- Added the `max_client_network_bandwidth` for `clickhouse-client` (Kirill Shvakov).

- Users with the `readonly = 2` setting are allowed to work with TEMPORARY tables (CREATE, DROP, INSERT...) (Kirill Shvakov).
- Added support for using multiple consumers with the `Kafka` engine. Extended configuration options for `Kafka` (Marek Vavruša).
- Added the `intExp3` and `intExp4` functions.
- Added the `sumKahan` aggregate function.
- Added the `to * Number*` `OrNull` functions, where `* Number*` is a numeric type.
- Added support for `WITH` clauses for an `INSERT SELECT` query (author: zhang2014).
- Added settings: `http_connection_timeout`, `http_send_timeout`, `http_receive_timeout`. In particular, these settings are used for downloading data parts for replication. Changing these settings allows for faster failover if the network is overloaded.
- Added support for `ALTER` for tables of type `Null` (Anastasiya Tsarkova).
- The `reinterpretAsString` function is extended for all data types that are stored contiguously in memory.
- Added the `--silent` option for the `clickhouse-local` tool. It suppresses printing query execution info in stderr.
- Added support for reading values of type `Date` from text in a format where the month and/or day of the month is specified using a single digit instead of two digits (Amos Bird).

## Performance Optimizations:

- Improved performance of aggregate functions `min`, `max`, `any`, `anyLast`, `anyHeavy`, `argMin`, `argMax` from string arguments.
- Improved performance of the functions `isInfinite`, `isFinite`, `isNaN`, `roundToExp2`.
- Improved performance of parsing and formatting `Date` and `DateTime` type values in text format.
- Improved performance and precision of parsing floating point numbers.
- Lowered memory usage for `JOIN` in the case when the left and right parts have columns with identical names that are not contained in `USING`.
- Improved performance of aggregate functions `varSamp`, `varPop`, `stddevSamp`, `stddevPop`, `covarSamp`, `covarPop`, `corr` by reducing computational stability. The old functions are available under the names `varSampStable`, `varPopStable`, `stddevSampStable`, `stddevPopStable`, `covarSampStable`, `covarPopStable`, `corrStable`.

## Bug Fixes:

- Fixed data deduplication after running a `DROP` or `DETACH PARTITION` query. In the previous version, dropping a partition and inserting the same data again was not working because inserted blocks were considered duplicates.
- Fixed a bug that could lead to incorrect interpretation of the `WHERE` clause for `CREATE MATERIALIZED VIEW` queries with `POPULATE`.
- Fixed a bug in using the `root_path` parameter in the `zookeeper_servers` configuration.
- Fixed unexpected results of passing the `Date` argument to `toStartOfDay`.
- Fixed the `addMonths` and `subtractMonths` functions and the arithmetic for `INTERVAL n MONTH` in cases when the result has the previous year.

- Added missing support for the `UUID` data type for `DISTINCT`, `JOIN`, and `uniq` aggregate functions and external dictionaries (Evgeniy Ivanov). Support for `UUID` is still incomplete.
- Fixed `SummingMergeTree` behavior in cases when the rows summed to zero.
- Various fixes for the `Kafka` engine (Marek Vavruša).
- Fixed incorrect behavior of the `Join` table engine (Amos Bird).
- Fixed incorrect allocator behavior under FreeBSD and OS X.
- The `extractAll` function now supports empty matches.
- Fixed an error that blocked usage of `libressl` instead of `openssl`.
- Fixed the `CREATE TABLE AS SELECT` query from temporary tables.
- Fixed non-atomicity of updating the replication queue. This could lead to replicas being out of sync until the server restarts.
- Fixed possible overflow in `gcd`, `lcm` and `modulo` (% operator) (Maks Skorokhod).
- `-preprocessed` files are now created after changing `umask` (`umask` can be changed in the config).
- Fixed a bug in the background check of parts (`MergeTreePartChecker`) when using a custom partition key.
- Fixed parsing of tuples (values of the `Tuple` data type) in text formats.
- Improved error messages about incompatible types passed to `multilif`, `array` and some other functions.
- Redesigned support for `Nullable` types. Fixed bugs that may lead to a server crash. Fixed almost all other bugs related to `NULL` support: incorrect type conversions in `INSERT SELECT`, insufficient support for `Nullable` in `HAVING` and `PREWHERE`, `join_use_nulls` mode, `Nullable` types as arguments of `OR` operator, etc.
- Fixed various bugs related to internal semantics of data types. Examples: unnecessary summing of `Enum` type fields in `SummingMergeTree`; alignment of `Enum` types in `Pretty` formats, etc.
- Stricter checks for allowed combinations of composite columns.
- Fixed the overflow when specifying a very large parameter for the `FixedString` data type.
- Fixed a bug in the `topK` aggregate function in a generic case.
- Added the missing check for equality of array sizes in arguments of n-ary variants of aggregate functions with an `-Array` combinator.
- Fixed a bug in `--pager` for `clickhouse-client` (author: ks1322).
- Fixed the precision of the `exp10` function.
- Fixed the behavior of the `visitParamExtract` function for better compliance with documentation.
- Fixed the crash when incorrect data types are specified.
- Fixed the behavior of `DISTINCT` in the case when all columns are constants.
- Fixed query formatting in the case of using the `tupleElement` function with a complex constant expression as the tuple element index.
- Fixed a bug in `Dictionary` tables for `range_hashed` dictionaries.
- Fixed a bug that leads to excessive rows in the result of `FULL` and `RIGHT JOIN` (Amos Bird).

- Fixed a server crash when creating and removing temporary files in `config.d` directories during config reload.
- Fixed the `SYSTEM DROP DNS CACHE` query: the cache was flushed but addresses of cluster nodes were not updated.
- Fixed the behavior of `MATERIALIZED VIEW` after executing `DETACH TABLE` for the table under the view (Marek Vavruša).

## Build Improvements:

- The `pbuilder` tool is used for builds. The build process is almost completely independent of the build host environment.
- A single build is used for different OS versions. Packages and binaries have been made compatible with a wide range of Linux systems.
- Added the `clickhouse-test` package. It can be used to run functional tests.
- The source tarball can now be published to the repository. It can be used to reproduce the build without using GitHub.
- Added limited integration with Travis CI. Due to limits on build time in Travis, only the debug build is tested and a limited subset of tests are run.
- Added support for `Cap'n'Proto` in the default build.
- Changed the format of documentation sources from `Restricted Text` to `Markdown`.
- Added support for `systemd` (Vladimir Smirnov). It is disabled by default due to incompatibility with some OS images and can be enabled manually.
- For dynamic code generation, `clang` and `lld` are embedded into the `clickhouse` binary. They can also be invoked as `clickhouse clang` and `clickhouse lld`.
- Removed usage of GNU extensions from the code. Enabled the `-Wextra` option. When building with `clang` the default is `libc++` instead of `libstdc++`.
- Extracted `clickhouse_parsers` and `clickhouse_common_io` libraries to speed up builds of various tools.

## Backward Incompatible Changes:

- The format for marks in `Log` type tables that contain `Nullable` columns was changed in a backward incompatible way. If you have these tables, you should convert them to the `TinyLog` type before starting up the new server version. To do this, replace `ENGINE = Log` with `ENGINE = TinyLog` in the corresponding `.sql` file in the `metadata` directory. If your table does not have `Nullable` columns or if the type of your table is not `Log`, then you do not need to do anything.
- Removed the `experimental_allow_extended_storage_definition_syntax` setting. Now this feature is enabled by default.
- The `runningIncome` function was renamed to `runningDifferenceStartingWithFirstValue` to avoid confusion.
- Removed the `FROM ARRAY JOIN arr` syntax when `ARRAY JOIN` is specified directly after `FROM` with no table (Amos Bird).
- Removed the `BlockTabSeparated` format that was used solely for demonstration purposes.

- Changed the state format for aggregate functions `varSamp`, `varPop`, `stddevSamp`, `stddevPop`, `covarSamp`, `covarPop`, `corr`. If you have stored states of these aggregate functions in tables (using the `AggregateFunction` data type or materialized views with corresponding states), please write to [clickhouse-feedback@yandex-team.com](mailto:clickhouse-feedback@yandex-team.com).
- In previous server versions there was an undocumented feature: if an aggregate function depends on parameters, you can still specify it without parameters in the `AggregateFunction` data type. Example: `AggregateFunction(quantiles, UInt64)` instead of `AggregateFunction(quantiles(0.5, 0.9), UInt64)`. This feature was lost. Although it was undocumented, we plan to support it again in future releases.
- Enum data types cannot be used in min/max aggregate functions. This ability will be returned in the next release.

#### Please Note When Upgrading:

- When doing a rolling update on a cluster, at the point when some of the replicas are running the old version of ClickHouse and some are running the new version, replication is temporarily stopped and the message `unknown parameter 'shard'` appears in the log. Replication will continue after all replicas of the cluster are updated.
- If different versions of ClickHouse are running on the cluster servers, it is possible that distributed queries using the following functions will have incorrect results: `varSamp`, `varPop`, `stddevSamp`, `stddevPop`, `covarSamp`, `covarPop`, `corr`. You should update all cluster nodes.

## Changelog for 2017

---

### ClickHouse Release 1.1.54327, 2017-12-21

This release contains bug fixes for the previous release 1.1.54318:

- Fixed bug with possible race condition in replication that could lead to data loss. This issue affects versions 1.1.54310 and 1.1.54318. If you use one of these versions with Replicated tables, the update is strongly recommended. This issue shows in logs in Warning messages like `Part ... from own log does not exist`. The issue is relevant even if you do not see these messages in logs.

### ClickHouse Release 1.1.54318, 2017-11-30

This release contains bug fixes for the previous release 1.1.54310:

- Fixed incorrect row deletions during merges in the SummingMergeTree engine
- Fixed a memory leak in unreplicated MergeTree engines
- Fixed performance degradation with frequent inserts in MergeTree engines
- Fixed an issue that was causing the replication queue to stop running
- Fixed rotation and archiving of server logs

### ClickHouse Release 1.1.54310, 2017-11-01

#### New Features:

- Custom partitioning key for the MergeTree family of table engines.
- **Kafka** table engine.
- Added support for loading **CatBoost** models and applying them to data stored in ClickHouse.
- Added support for time zones with non-integer offsets from UTC.

- Added support for arithmetic operations with time intervals.
- The range of values for the Date and DateTime types is extended to the year 2105.
- Added the CREATE MATERIALIZED VIEW x TO y query (specifies an existing table for storing the data of a materialized view).
- Added the `ATTACH TABLE` query without arguments.
- The processing logic for Nested columns with names ending in -Map in a SummingMergeTree table was extracted to the sumMap aggregate function. You can now specify such columns explicitly.
- Max size of the IP trie dictionary is increased to 128M entries.
- Added the `getSizeOfEnumType` function.
- Added the `sumWithOverflow` aggregate function.
- Added support for the Cap'n Proto input format.
- You can now customize compression level when using the zstd algorithm.

#### Backward Incompatible Changes:

- Creation of temporary tables with an engine other than Memory is not allowed.
- Explicit creation of tables with the View or MaterializedView engine is not allowed.
- During table creation, a new check verifies that the sampling key expression is included in the primary key.

#### Bug Fixes:

- Fixed hangups when synchronously inserting into a Distributed table.
- Fixed nonatomic adding and removing of parts in Replicated tables.
- Data inserted into a materialized view is not subjected to unnecessary deduplication.
- Executing a query to a Distributed table for which the local replica is lagging and remote replicas are unavailable does not result in an error anymore.
- Users do not need access permissions to the `default` database to create temporary tables anymore.
- Fixed crashing when specifying the Array type without arguments.
- Fixed hangups when the disk volume containing server logs is full.
- Fixed an overflow in the `toRelativeWeekNum` function for the first week of the Unix epoch.

#### Build Improvements:

- Several third-party libraries (notably Poco) were updated and converted to git submodules.

## ClickHouse Release 1.1.54304, 2017-10-19

#### New Features:

- TLS support in the native protocol (to enable, set `tcp_ssl_port` in `config.xml` ).

#### Bug Fixes:

- `ALTER` for replicated tables now tries to start running as soon as possible.
- Fixed crashing when reading data with the setting `preferred_block_size_bytes=0`.

- Fixed crashes of `clickhouse-client` when pressing `Page Down`
- Correct interpretation of certain complex queries with `GLOBAL IN` and `UNION ALL`
- `FREEZE PARTITION` always works atomically now.
- Empty POST requests now return a response with code 411.
- Fixed interpretation errors for expressions like `CAST(1 AS Nullable(UInt8))`.
- Fixed an error when reading `Array(Nullable(String))` columns from `MergeTree` tables.
- Fixed crashing when parsing queries like `SELECT dummy AS dummy, dummy AS b`
- Users are updated correctly with invalid `users.xml`
- Correct handling when an executable dictionary returns a non-zero response code.

## ClickHouse Release 1.1.54292, 2017-09-20

### New Features:

- Added the `pointInPolygon` function for working with coordinates on a coordinate plane.
- Added the `sumMap` aggregate function for calculating the sum of arrays, similar to `SummingMergeTree`.
- Added the `trunc` function. Improved performance of the rounding functions (`round`, `floor`, `ceil`, `roundToExp2`) and corrected the logic of how they work. Changed the logic of the `roundToExp2` function for fractions and negative numbers.
- The ClickHouse executable file is now less dependent on the `libc` version. The same ClickHouse executable file can run on a wide variety of Linux systems. There is still a dependency when using compiled queries (with the setting `compile = 1`, which is not used by default).
- Reduced the time needed for dynamic compilation of queries.

### Bug Fixes:

- Fixed an error that sometimes produced `part ... intersects previous part` messages and weakened replica consistency.
- Fixed an error that caused the server to lock up if ZooKeeper was unavailable during shutdown.
- Removed excessive logging when restoring replicas.
- Fixed an error in the `UNION ALL` implementation.
- Fixed an error in the `concat` function that occurred if the first column in a block has the `Array` type.
- Progress is now displayed correctly in the `system.merges` table.

## ClickHouse Release 1.1.54289, 2017-09-13

### New Features:

- `SYSTEM` queries for server administration: `SYSTEM RELOAD DICTIONARY`, `SYSTEM RELOAD DICTIONARIES`, `SYSTEM DROP DNS CACHE`, `SYSTEM SHUTDOWN`, `SYSTEM KILL`.
- Added functions for working with arrays: `concat`, `arraySlice`, `arrayPushBack`, `arrayPushFront`, `arrayPopBack`, `arrayPopFront`.
- Added `root` and `identity` parameters for the ZooKeeper configuration. This allows you to isolate individual users on the same ZooKeeper cluster.

- Added aggregate functions `groupBitAnd`, `groupBitOr`, and `groupBitXor` (for compatibility, they are also available under the names `BIT_AND`, `BIT_OR`, and `BIT_XOR`).
- External dictionaries can be loaded from MySQL by specifying a socket in the filesystem.
- External dictionaries can be loaded from MySQL over SSL (`ssl_cert`, `ssl_key`, `ssl_ca` parameters).
- Added the `max_network_bandwidth_for_user` setting to restrict the overall bandwidth use for queries per user.
- Support for `DROP TABLE` for temporary tables.
- Support for reading `DateTime` values in Unix timestamp format from the `CSV` and `JSONEachRow` formats.
- Lagging replicas in distributed queries are now excluded by default (the default threshold is 5 minutes).
- FIFO locking is used during ALTER: an ALTER query isn't blocked indefinitely for continuously running queries.
- Option to set `umask` in the config file.
- Improved performance for queries with `DISTINCT`.

#### Bug Fixes:

- Improved the process for deleting old nodes in ZooKeeper. Previously, old nodes sometimes didn't get deleted if there were very frequent inserts, which caused the server to be slow to shut down, among other things.
- Fixed randomization when choosing hosts for the connection to ZooKeeper.
- Fixed the exclusion of lagging replicas in distributed queries if the replica is localhost.
- Fixed an error where a data part in a `ReplicatedMergeTree` table could be broken after running `ALTER MODIFY` on an element in a `Nested` structure.
- Fixed an error that could cause `SELECT` queries to "hang".
- Improvements to distributed DDL queries.
- Fixed the query `CREATE TABLE ... AS <materialized view>`.
- Resolved the deadlock in the `ALTER ... CLEAR COLUMN IN PARTITION` query for `Buffer` tables.
- Fixed the invalid default value for `Enum` s (0 instead of the minimum) when using the `JSONEachRow` and `TSKV` formats.
- Resolved the appearance of zombie processes when using a dictionary with an `executable` source.
- Fixed segfault for the `HEAD` query.

#### Improved Workflow for Developing and Assembling ClickHouse:

- You can use `pbuilder` to build ClickHouse.
- You can use `libc++` instead of `libstdc++` for builds on Linux.
- Added instructions for using static code analysis tools: `Coverage`, `clang-tidy`, `cppcheck`.

#### Please Note When Upgrading:

- There is now a higher default value for the MergeTree setting `max_bytes_to_merge_at_max_space_in_pool` (the maximum total size of data parts to merge, in bytes): it has increased from 100 GiB to 150 GiB. This might result in large merges running after the server upgrade, which could cause an increased load on the disk subsystem. If the free space available on the server is less than twice the total amount of the merges that are running, this will cause all other merges to stop running, including merges of small data parts. As a result, INSERT queries will fail with the message “Merges are processing significantly slower than inserts.” Use the `SELECT * FROM system.merges` query to monitor the situation. You can also check the `DiskSpaceReservedForMerge` metric in the `system.metrics` table, or in Graphite. You do not need to do anything to fix this, since the issue will resolve itself once the large merges finish. If you find this unacceptable, you can restore the previous value for the `max_bytes_to_merge_at_max_space_in_pool` setting. To do this, go to the `<merge_tree>` section in `config.xml`, set `<merge_tree>``<max_bytes_to_merge_at_max_space_in_pool>107374182400</max_bytes_to_merge_at_max_space_in_pool>` and restart the server.

## ClickHouse Release 1.1.54284, 2017-08-29

- This is a bugfix release for the previous 1.1.54282 release. It fixes leaks in the parts directory in ZooKeeper.

## ClickHouse Release 1.1.54282, 2017-08-23

This release contains bug fixes for the previous release 1.1.54276:

- Fixed DB::Exception: Assertion violation: `!_path.empty()` when inserting into a Distributed table.
- Fixed parsing when inserting in RowBinary format if input data starts with';'.
- Errors during runtime compilation of certain aggregate functions (e.g. `groupArray()`).

## ClickHouse Release 1.1.54276, 2017-08-16

### New Features:

- Added an optional WITH section for a SELECT query. Example query: `WITH 1+1 AS a SELECT a, a*a`
- INSERT can be performed synchronously in a Distributed table: OK is returned only after all the data is saved on all the shards. This is activated by the setting `insert_distributed_sync=1`.
- Added the UUID data type for working with 16-byte identifiers.
- Added aliases of CHAR, FLOAT and other types for compatibility with the Tableau.
- Added the functions `toYYYYMM`, `toYYYYMMDD`, and `toYYYYMMDDhhmmss` for converting time into numbers.
- You can use IP addresses (together with the hostname) to identify servers for clustered DDL queries.
- Added support for non-constant arguments and negative offsets in the function `substring(str, pos, len)`.
- Added the `max_size` parameter for the `groupArray(max_size)(column)` aggregate function, and optimized its performance.

### Main Changes:

- Security improvements: all server files are created with 0640 permissions (can be changed via `<umask>` config parameter).
- Improved error messages for queries with invalid syntax.
- Significantly reduced memory consumption and improved performance when merging large sections of MergeTree data.

- Significantly increased the performance of data merges for the ReplacingMergeTree engine.
- Improved performance for asynchronous inserts from a Distributed table by combining multiple source inserts. To enable this functionality, use the setting `distributed_directory_monitor_batch_inserts=1`.

## Backward Incompatible Changes:

- Changed the binary format of aggregate states of `groupArray(array_column)` functions for arrays.

## Complete List of Changes:

- Added the `output_format_json_quote_denormals` setting, which enables outputting nan and inf values in JSON format.
- Optimized stream allocation when reading from a Distributed table.
- Settings can be configured in readonly mode if the value does not change.
- Added the ability to retrieve non-integer granules of the MergeTree engine in order to meet restrictions on the block size specified in the `preferred_block_size_bytes` setting. The purpose is to reduce the consumption of RAM and increase cache locality when processing queries from tables with large columns.
- Efficient use of indexes that contain expressions like `toStartOfHour(x)` for conditions like `toStartOfHour(x) op constexpr`.
- Added new settings for MergeTree engines (the `merge_tree` section in `config.xml`):
  - `replicated_deduplication_window_seconds` sets the number of seconds allowed for deduplicating inserts in Replicated tables.
  - `cleanup_delay_period` sets how often to start cleanup to remove outdated data.
  - `replicated_can_become_leader` can prevent a replica from becoming the leader (and assigning merges).
- Accelerated cleanup to remove outdated data from ZooKeeper.
- Multiple improvements and fixes for clustered DDL queries. Of particular interest is the new setting `distributed_ddl_task_timeout`, which limits the time to wait for a response from the servers in the cluster. If a ddl request has not been performed on all hosts, a response will contain a timeout error and a request will be executed in an async mode.
- Improved display of stack traces in the server logs.
- Added the “none” value for the compression method.
- You can use multiple `dictionaries_config` sections in `config.xml`.
- It is possible to connect to MySQL through a socket in the file system.
- The `system.parts` table has a new column with information about the size of marks, in bytes.

## Bug Fixes:

- Distributed tables using a Merge table now work correctly for a SELECT query with a condition on the `_table` field.
- Fixed a rare race condition in ReplicatedMergeTree when checking data parts.
- Fixed possible freezing on “leader election” when starting a server.
- The `max_replica_delay_for_distributed_queries` setting was ignored when using a local replica of the data source. This has been fixed.

- Fixed incorrect behavior of `ALTER TABLE CLEAR COLUMN IN PARTITION` when attempting to clean a non-existing column.
- Fixed an exception in the `multilf` function when using empty arrays or strings.
- Fixed excessive memory allocations when deserializing Native format.
- Fixed incorrect auto-update of Trie dictionaries.
- Fixed an exception when running queries with a GROUP BY clause from a Merge table when using SAMPLE.
- Fixed a crash of GROUP BY when using `distributed_aggregation_memory_efficient=1`.
- Now you can specify the `database.table` in the right side of IN and JOIN.
- Too many threads were used for parallel aggregation. This has been fixed.
- Fixed how the “if” function works with `FixedString` arguments.
- SELECT worked incorrectly from a Distributed table for shards with a weight of 0. This has been fixed.
- Running `CREATE VIEW IF EXISTS` no longer causes crashes.
- Fixed incorrect behavior when `input_format_skip_unknown_fields=1` is set and there are negative numbers.
- Fixed an infinite loop in the `dictGetHierarchy()` function if there is some invalid data in the dictionary.
- Fixed Syntax error: unexpected (...) errors when running distributed queries with subqueries in an IN or JOIN clause and Merge tables.
- Fixed an incorrect interpretation of a SELECT query from Dictionary tables.
- Fixed the “Cannot mremap” error when using arrays in IN and JOIN clauses with more than 2 billion elements.
- Fixed the failover for dictionaries with MySQL as the source.

#### Improved Workflow for Developing and Assembling ClickHouse:

- Builds can be assembled in Arcadia.
- You can use gcc 7 to compile ClickHouse.
- Parallel builds using `ccache+distcc` are faster now.

## ClickHouse Release 1.1.54245, 2017-07-04

#### New Features:

- Distributed DDL (for example, `CREATE TABLE ON CLUSTER`)
- The replicated query `ALTER TABLE CLEAR COLUMN IN PARTITION`.
- The engine for Dictionary tables (access to dictionary data in the form of a table).
- Dictionary database engine (this type of database automatically has Dictionary tables available for all the connected external dictionaries).
- You can check for updates to the dictionary by sending a request to the source.
- Qualified column names

- Quoting identifiers using double quotation marks.
- Sessions in the HTTP interface.
- The OPTIMIZE query for a Replicated table can run not only on the leader.

### Backward Incompatible Changes:

- Removed SET GLOBAL.

### Minor Changes:

- Now after an alert is triggered, the log prints the full stack trace.
- Relaxed the verification of the number of damaged/extraneous data parts at startup (there were too many false positives).

### Bug Fixes:

- Fixed a bad connection “sticking” when inserting into a Distributed table.
- GLOBAL IN now works for a query from a Merge table that looks at a Distributed table.
- The incorrect number of cores was detected on a Google Compute Engine virtual machine. This has been fixed.
- Changes in how an executable source of cached external dictionaries works.
- Fixed the comparison of strings containing null characters.
- Fixed the comparison of Float32 primary key fields with constants.
- Previously, an incorrect estimate of the size of a field could lead to overly large allocations.
- Fixed a crash when querying a Nullable column added to a table using ALTER.
- Fixed a crash when sorting by a Nullable column, if the number of rows is less than LIMIT.
- Fixed an ORDER BY subquery consisting of only constant values.
- Previously, a Replicated table could remain in the invalid state after a failed DROP TABLE.
- Aliases for scalar subqueries with empty results are no longer lost.
- Now a query that used compilation does not fail with an error if the .so file gets damaged.

---

## Roadmap

The roadmap for the year 2021 is published for open discussion [here](#).

---

## Fixed in ClickHouse 21.4.3.21, 2021-04-12

### CVE-2021-25263

An attacker that has CREATE DICTIONARY privilege, can read arbitrary file outside permitted directory.

Fix has been pushed to versions 20.8.18.32-Its, 21.1.9.41-stable, 21.2.9.41-stable, 21.3.6.55-Its, 21.4.3.21-stable and later.

Credits: [Vyacheslav Egoshin](#)

## Fixed in ClickHouse Release 19.14.3.3, 2019-09-10

### CVE-2019-15024

An attacker that has write access to ZooKeeper and who can run a custom server available from the network where ClickHouse runs, can create a custom-built malicious server that will act as a ClickHouse replica and register it in ZooKeeper. When another replica will fetch data part from the malicious replica, it can force clickhouse-server to write to arbitrary path on filesystem.

Credits: Eldar Zaitov of Yandex Information Security Team

### CVE-2019-16535

An OOB read, OOB write and integer underflow in decompression algorithms can be used to achieve RCE or DoS via native protocol.

Credits: Eldar Zaitov of Yandex Information Security Team

### CVE-2019-16536

Stack overflow leading to DoS can be triggered by a malicious authenticated client.

Credits: Eldar Zaitov of Yandex Information Security Team

## Fixed in ClickHouse Release 19.13.6.1, 2019-09-20

### CVE-2019-18657

Table function `url` had the vulnerability allowed the attacker to inject arbitrary HTTP headers in the request.

Credits: [Nikita Tikhomirov](#)

## Fixed in ClickHouse Release 18.12.13, 2018-09-10

### CVE-2018-14672

Functions for loading CatBoost models allowed path traversal and reading arbitrary files through error messages.

Credits: Andrey Krasichkov of Yandex Information Security Team

## Fixed in ClickHouse Release 18.10.3, 2018-08-13

### CVE-2018-14671

unixODBC allowed loading arbitrary shared objects from the file system which led to a Remote Code Execution vulnerability.

Credits: Andrey Krasichkov and Evgeny Sidorov of Yandex Information Security Team

## Fixed in ClickHouse Release 1.1.54388, 2018-06-28

### CVE-2018-14668

“remote” table function allowed arbitrary symbols in “user”, “password” and “default\_database” fields which led to Cross Protocol Request Forgery Attacks.

Credits: Andrey Krasichkov of Yandex Information Security Team

## Fixed in ClickHouse Release 1.1.54390, 2018-07-06

## CVE-2018-14669

ClickHouse MySQL client had “LOAD DATA LOCAL INFILE” functionality enabled that allowed a malicious MySQL database read arbitrary files from the connected ClickHouse server.

Credits: Andrey Krasichkov and Evgeny Sidorov of Yandex Information Security Team

Fixed in ClickHouse Release 1.1.54131, 2017-01-10

## CVE-2018-14670

Incorrect configuration in deb package could lead to the unauthorized use of the database.

Credits: the UK’s National Cyber Security Centre (NCSC)

## What’s New in ClickHouse?

There’s a short high-level [roadmap](#) and a detailed [changelog](#) for releases that have already been published.

**Was this content helpful?**

**RATING STARS**

Rating: RATING\_VALUE - RATING\_COUNT

votes

©2016-2021 ClickHouse, Inc.

Built from c936fa93c3