

# Что такое ClickHouse

ClickHouse - столбцовая система управления базами данных (СУБД) для онлайн обработки аналитических запросов (OLAP).

В обычной, «строковой» СУБД, данные хранятся в таком порядке:

Строка	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	...	...	...	...	...

То есть, значения, относящиеся к одной строке, физически хранятся рядом.

Примеры строковых СУБД: MySQL, Postgres, MS SQL Server.

В столбцовых СУБД, данные хранятся в таком порядке:

Строка:	#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	...

В примерах изображён только порядок расположения данных.

То есть, значения из разных столбцов хранятся отдельно, а данные одного столбца - вместе.

Примеры столбцовых СУБД: Vertica, Paraccel (Actian Matrix, Amazon Redshift), Sybase IQ, Exasol, Infobright, InfiniDB, MonetDB (VectorWise, Actian Vector), LucidDB, SAP HANA, Google Dremel, Google PowerDrill, Druid, kdb+.

Разный порядок хранения данных лучше подходит для разных сценариев работы.

Сценарий работы с данными - это то, какие производятся запросы, как часто и в каком соотношении; сколько читается данных на запросы каждого вида - строк, столбцов, байт; как соотносятся чтения и обновления данных; какой рабочий размер данных и насколько локально он используется; используются ли транзакции и с какой изолированностью; какие требования к дублированию данных и логической целостности; требования к задержкам на выполнение и пропускной способности запросов каждого вида и т. п.

Чем больше нагрузка на систему, тем более важной становится специализация под сценарий работы, и тем более конкретной становится эта специализация. Не существует системы, одинаково хорошо подходящей под существенно различные сценарии работы. Если система подходит под широкое множество сценариев работы, то при достаточно большой нагрузке, система будет справляться со всеми сценариями работы плохо, или справляться хорошо только с одним из сценариев работы.

## Ключевые особенности OLAP сценария работы

- подавляющее большинство запросов - на чтение;
- данные обновляются достаточно большими пачками (> 1000 строк), а не по одной строке, или не обновляются вообще;
- данные добавляются в БД, но не изменяются;
- при чтении, вынимается достаточно большое количество строк из БД, но только небольшое подмножество столбцов;
- таблицы являются «широкими», то есть, содержат большое количество столбцов;
- запросы идут сравнительно редко (обычно не более сотни в секунду на сервер);
- при выполнении простых запросов, допустимы задержки в районе 50 мс;
- значения в столбцах достаточно мелкие - числа и небольшие строки (пример - 60 байт на URL);
- требуется высокая пропускная способность при обработке одного запроса (до миллиардов строк в секунду на один сервер);
- транзакции отсутствуют;
- низкие требования к консистентности данных;
- в запросе одна большая таблица, все таблицы кроме одной маленькие;
- результат выполнения запроса существенно меньше исходных данных - то есть, данные фильтруются или агрегируются; результат выполнения помещается в оперативку на одном сервере.

Легко видеть, что OLAP сценарий работы существенно отличается от других распространённых сценариев работы (например, OLTP или Key-Value сценариев работы). Таким образом, не имеет никакого смысла пытаться использовать OLTP или Key-Value БД для обработки аналитических запросов, если вы хотите получить приличную производительность («выше плинтуса»). Например, если вы попытаетесь использовать для аналитики MongoDB или Redis - вы получите анекдотически низкую производительность по сравнению с OLAP-СУБД.

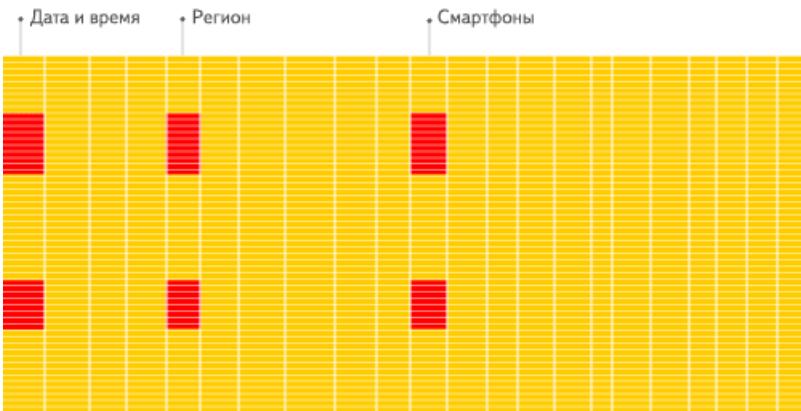
## Причины, по которым столбцовые СУБД лучше подходят для OLAP сценария

Столбцовые СУБД лучше (от 100 раз по скорости обработки большинства запросов) подходят для OLAP сценария работы. Причины в деталях будут разъяснены ниже, а сам факт проще продемонстрировать визуально:

### Строковые СУБД



## Столбцовые СУБД



Видите разницу?

## По вводу-выводу

1. Для выполнения аналитического запроса, требуется прочитать небольшое количество столбцов таблицы. В столбцовой БД для этого можно читать только нужные данные. Например, если вам требуется только 5 столбцов из 100, то следует рассчитывать на 20-кратное уменьшение ввода-вывода.
2. Так как данные читаются пачками, то их проще сжимать. Данные, лежащие по столбцам также лучше сжимаются. За счёт этого, дополнительно уменьшается объём ввода-вывода.
3. За счёт уменьшения ввода-вывода, больше данных влезает в системный кэш.

Например, для запроса «посчитать количество записей для каждой рекламной системы», требуется прочитать один столбец «идентификатор рекламной системы», который занимает 1 байт в несжатом виде. Если большинство переходов было не с рекламных систем, то можно рассчитывать хотя бы на десятикратное сжатие этого столбца. При использовании быстрого алгоритма сжатия, возможно разжатие данных со скоростью более нескольких гигабайт несжатых данных в секунду. То есть, такой запрос может выполняться со скоростью около нескольких миллиардов строк в секунду на одном сервере. На практике, такая скорость действительно достигается.

## По вычислениям

Так как для выполнения запроса надо обработать достаточно большое количество строк, становится актуальным диспетчеризовывать все операции не для отдельных строк, а для целых векторов, или реализовать движок выполнения запроса так, чтобы издержки на диспетчеризацию были примерно нулевыми. Если этого не делать, то при любой не слишком плохой дисковой подсистеме, интерпретатор запроса неизбежно упрётся в CPU.

Имеет смысл не только хранить данные по столбцам, но и обрабатывать их, по возможности, тоже по столбцам.

Есть два способа это сделать:

1. Векторный движок. Все операции пишутся не для отдельных значений, а для векторов. То есть, вызывать операции надо достаточно редко, и издержки на диспетчеризацию становятся пренебрежимо маленькими. Код операции содержит в себе хорошо оптимизированный внутренний цикл.
2. Кодогенерация. Для запроса генерируется код, в котором подставлены все косвенные вызовы.

В «обычных» БД этого не делается, так как не имеет смысла при выполнении простых запросов. Хотя есть исключения. Например, в MemSQL кодогенерация используется для уменьшения latency при выполнении SQL запросов. Для сравнения, в аналитических СУБД требуется оптимизация throughput, а не latency.

Стоит заметить, что для эффективности по CPU требуется, чтобы язык запросов был декларативным (SQL, MDX) или хотя бы векторным (J, K). То есть, чтобы запрос содержал циклы только в неявном виде, открывая возможности для оптимизации.

## Отличительные возможности ClickHouse

### По-настоящему столбцовая СУБД

В по-настоящему столбцовой СУБД рядом со значениями не хранится никаких лишних данных. Например, должны поддерживаться значения постоянной длины, чтобы не хранить рядом со значениями типа «число» их длины. Для примера, миллиард значений типа UInt8 должен действительно занимать в несжатом виде около 1GB, иначе это сильно ударит по эффективности использования CPU. Очень важно хранить данные компактно (без «мусора») в том числе в несжатом виде, так как скорость разжатия (использование CPU) зависит, в основном, от объёма несжатых данных.

Этот пункт пришлось выделить, так как существуют системы, которые могут хранить значения отдельных столбцов по отдельности, но не могут эффективно выполнять аналитические запросы в силу оптимизации под другой сценарий работы. Примеры: HBase, BigTable, Cassandra, HyperTable. В этих системах вы получите пропускную способность в районе сотен тысяч строк в секунду, но не сотен миллионов строк в секунду.

Также стоит заметить, что ClickHouse является системой управления базами данных, а не одной базой данных. То есть, ClickHouse позволяет создавать таблицы и базы данных в runtime, загружать данные и выполнять запросы без переконфигурирования и перезапуска сервера.

### Сжатие данных

Некоторые столбцовые СУБД (InfiniDB CE, MonetDB) не используют сжатие данных. Однако сжатие данных действительно играет одну из ключевых ролей в демонстрации отличной производительности.

### Хранение данных на диске

Многие столбцовые СУБД (SAP HANA, Google PowerDrill) могут работать только в оперативной памяти. Такой подход стимулирует выделять больший бюджет на оборудование, чем фактически требуется для анализа в реальном времени. ClickHouse спроектирован для работы на обычных жестких дисках, что обеспечивает низкую стоимость хранения на гигабайт данных, но SSD и дополнительная оперативная память тоже полноценно используются, если доступны.

### Параллельная обработка запроса на многих процессорных ядрах

Большие запросы естественным образом распараллеливаются, используя все необходимые ресурсы из доступных на сервере.

## Распределённая обработка запроса на многих серверах

Почти все перечисленные ранее столбцовые СУБД не поддерживают распределённую обработку запроса.

В ClickHouse данные могут быть расположены на разных шардах. Каждый шард может представлять собой группу реплик, которые используются для отказоустойчивости. Запрос будет выполнен на всех шардах параллельно. Это делается прозрачно для пользователя.

## Поддержка SQL

ClickHouse поддерживает декларативный язык запросов на основе SQL и во многих случаях совпадающий с SQL стандартом.

Поддерживаются GROUP BY, ORDER BY, подзапросы в секциях FROM, IN, JOIN, а также скалярные подзапросы.

Зависимые подзапросы и оконные функции не поддерживаются.

## Векторный движок

Данные не только хранятся по столбцам, но и обрабатываются по векторам - кусочкам столбцов. За счёт этого достигается высокая эффективность по CPU.

## Обновление данных в реальном времени

ClickHouse поддерживает таблицы с первичным ключом. Для того, чтобы можно было быстро выполнять запросы по диапазону первичного ключа, данные инкрементально сортируются с помощью merge дерева. За счёт этого, поддерживается постоянное добавление данных в таблицу. Блокировки при добавлении данных отсутствуют.

## Наличие индекса

Физическая сортировка данных по первичному ключу позволяет получать данные для конкретных его значений или их диапазонов с низкими задержками - менее десятков миллисекунд.

## Подходит для онлайн запросов

Низкие задержки позволяют не откладывать выполнение запроса и не подготавливать ответ заранее, а выполнять его именно в момент загрузки страницы пользовательского интерфейса. То есть, в режиме онлайн.

## Поддержка приближённых вычислений

ClickHouse предоставляет различные способы разменять точность вычислений на производительность:

1. Система содержит агрегатные функции для приближённого вычисления количества различных значений, медианы и квантилей.
2. Поддерживается возможность выполнить запрос на основе части (выборки) данных и получить приближённый результат. При этом, с диска будет считано пропорционально меньше данных.
3. Поддерживается возможность выполнить агрегацию не для всех ключей, а для ограниченного количества первых попавшихся ключей. При выполнении некоторых условий на распределение ключей в данных, это позволяет получить достаточно точный результат с использованием меньшего количества ресурсов.

## Репликация данных и поддержка целостности

Используется асинхронная multimeter репликация. После записи на любую доступную реплику, данные распространяются на все остальные реплики в фоне. Система поддерживает полную идентичность данных на разных репликах. Восстановление после большинства сбоев осуществляется автоматически, а в сложных случаях — полуавтоматически. При необходимости, можно [включить кворумную запись](#) данных.

Подробнее смотрите раздел [Репликация данных](#).

## Особенности, которые могут считаться недостатками

1. Отсутствие полноценных транзакций.
2. Возможность изменять или удалять ранее записанные данные с низкими задержками и высокой частотой запросов не предоставляется. Есть массовое удаление и изменение данных для очистки более не нужного или соответствия [GDPR](#).
3. Разреженный индекс делает ClickHouse плохо пригодным для точечных чтений одиночных строк по своим ключам.

---

## Производительность

По результатам внутреннего тестирования в Яндексе, ClickHouse обладает наиболее высокой производительностью (как наиболее высокой пропускной способностью на длинных запросах, так и наиболее низкой задержкой на коротких запросах), при соответствующем сценарии работы, среди доступных для тестирования систем подобного класса. Результаты тестирования можно посмотреть на [отдельной странице](#).

Также это подтверждают многочисленные независимые бенчмарки. Их не сложно найти в Интернете самостоятельно, либо можно воспользоваться [небольшой коллекцией ссылок по теме](#).

## Пропускная способность при обработке одного большого запроса

Пропускную способность можно измерять в строчках в секунду и в мегабайтах в секунду. При условии, что данные помещаются в page cache, не слишком сложный запрос обрабатывается на современном железе со скоростью около 2-10 GB/sec. несжатых данных на одном сервере (в простейшем случае скорость может достигать 30 GB/sec). Если данные не помещаются в page cache, то скорость работы зависит от скорости дисковой подсистемы и коэффициента сжатия данных. Например, если дисковая подсистема позволяет читать данные со скоростью 400 MB/sec., а коэффициент сжатия данных составляет 3, то скорость будет около 1.2GB/sec. Для получения скорости в строчках в секунду, следует поделить скорость в байтах в секунду на суммарный размер используемых в запросе столбцов. Например, если вынимаются столбцы на 10 байт, то скорость будет в районе 100-200 млн. строк в секунду.

При распределённой обработке запроса, скорость обработки запроса растёт почти линейно, но только при условии, что в результате агрегации или при сортировке получается не слишком большое множество строчек.

## Задержки при обработке коротких запросов

Если запрос использует первичный ключ, и выбирает для обработки не слишком большое количество строчек (сотни тысяч), и использует не слишком большое количество столбцов, то вы можете рассчитывать на latency менее 50 миллисекунд (от единиц миллисекунд в лучшем случае), при условии, что данные помещаются в page cache. Иначе latency вычисляется из количества seek-

ов. Если вы используете вращающиеся диски, то на не слишком сильно нагруженной системе, latency вычисляется по формуле: seek time (10 мс.) \* количество столбцов в запросе \* количество кусков с данными.

## Пропускная способность при обработке многочисленных коротких запросов

При тех же условиях, ClickHouse может обработать несколько сотен (до нескольких тысяч в лучшем случае) запросов в секунду на одном сервере. Так как такой сценарий работы не является типичным для аналитических СУБД, рекомендуется рассчитывать не более чем на 100 запросов в секунду.

## Производительность при вставке данных

Данные рекомендуется вставлять пачками не менее 1000 строк или не более одного запроса в секунду. При вставке в таблицу типа MergeTree из tab-separated дампа, скорость вставки будет в районе 50-200 МБ/сек. Если вставляются строчки размером около 1 КБ, то скорость будет в районе 50 000 - 200 000 строчек в секунду. Если строчки маленькие - производительность в строчках в секунду будет выше (на данных БК - > 500 000 строк в секунду, на данных Graphite - > 1 000 000 строк в секунду). Для увеличения производительности, можно производить несколько запросов INSERT параллельно - при этом производительность растёт линейно.

## История ClickHouse

ClickHouse изначально разрабатывался для обеспечения работы Яндекс.Метрики, второй крупнейшей в мире платформы для веб аналитики, и продолжает быть её ключевым компонентом. При более 13 триллионах записей в базе данных и более 20 миллиардах событий в сутки, ClickHouse позволяет генерировать индивидуально настроенные отчёты на лету напрямую из неагрегированных данных. Данная статья вкратце демонстрирует какие цели исторически стояли перед ClickHouse на ранних этапах его развития.

Яндекс.Метрика на лету строит индивидуальные отчёты на основе хитов и визитов, с периодом и произвольными сегментами, задаваемыми конечным пользователем. Часто требуется построение сложных агрегатов, например числа уникальных пользователей. Новые данные для построения отчета поступают в реальном времени.

На апрель 2014, в Яндекс.Метрику поступало около 12 миллиардов событий (показов страниц и кликов мыши) ежедневно. Все эти события должны быть сохранены для возможности строить произвольные отчёты. Один запрос может потребовать просканировать миллионы строк за время не более нескольких сотен миллисекунд, или сотни миллионов строк за время не более нескольких секунд.

## Использование в Яндекс.Метрике и других отделах Яндекса

В Яндекс.Метрике ClickHouse используется для нескольких задач.

Основная задача - построение отчётов в режиме онлайн по неагрегированным данным. Для решения этой задачи используется кластер из 374 серверов, хранящий более 20,3 трллионов строк в базе данных. Объём сжатых данных, без учёта дублирования и репликации, составляет около 2 ПБ. Объём несжатых данных (в формате tsv) составил бы, приблизительно, 17 ПБ.

Также ClickHouse используется:

- для хранения данных Вебвизора;
- для обработки промежуточных данных;

- для построения глобальных отчётов Аналитиками;
- для выполнения запросов в целях отладки движка Метрики;
- для анализа логов работы API и пользовательского интерфейса.

ClickHouse имеет более десятка инсталляций в других отделах Яндекса: в Вертикальных сервисах, Маркете, Директе, БК, Бизнес аналитике, Мобильной разработке, AdFox, Персональных сервисах и т.п.

## Агрегированные и неагрегированные данные

Существует мнение, что для того, чтобы эффективно считать статистику, данные нужно агрегировать, так как это позволяет уменьшить объём данных.

Но агрегированные данные являются очень ограниченным решением, по следующим причинам:

- вы должны заранее знать перечень отчётов, необходимых пользователю;
- то есть, пользователь не может построить произвольный отчёт;
- при агрегации по большому количеству ключей, объём данных не уменьшается и агрегация бесполезна;
- при большом количестве отчётов, получается слишком много вариантов агрегации (комбинаторный взрыв);
- при агрегации по ключам высокой кардинальности (например, URL) объём данных уменьшается не сильно (менее чем в 2 раза);
- из-за этого, объём данных при агрегации может не уменьшиться, а вырасти;
- пользователи будут смотреть не все отчёты, которые мы для них посчитаем - то есть, большая часть вычислений бесполезна;
- возможно нарушение логической целостности данных для разных агрегаций;

Как видно, если ничего не агрегировать, и работать с неагрегированными данными, то это даже может уменьшить объём вычислений.

Впрочем, при агрегации, существенная часть работы выносится в оффлайне, и её можно делать сравнительно спокойно. Для сравнения, при онлайн вычислениях, вычисления надо делать так быстро, как это возможно, так как именно в момент вычислений пользователь ждёт результата.

В Яндекс.Метрике есть специализированная система для агрегированных данных - Metrage, на основе которой работает большинство отчётов.

Также в Яндекс.Метрике с 2009 года использовалась специализированная OLAP БД для неагрегированных данных - OLAPServer, на основе которой раньше работал конструктор отчётов. OLAPServer хорошо подходил для неагрегированных данных, но содержал много ограничений, не позволяющих использовать его для всех отчётов так, как хочется: отсутствие поддержки типов данных (только числа), невозможность инкрементального обновления данных в реальном времени (только перезаписью данных за сутки). OLAPServer не является СУБД, а является специализированной БД.

Чтобы снять ограничения OLAPServer-а и решить задачу работы с неагрегированными данными для всех отчётов, разработана СУБД ClickHouse.

## 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* (of single replica)	Ref
2gis	Maps	Monitoring	—	—	Talk i July 2
Admiral	Martech	Engagement Management	—	—	Webinar June 1
AdScribe	Ads	TV Analytics	—	—	A quick
Ahrefs	SEO	Analytics	—	—	Job lis
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

Company	Industry	Usecase	Cluster Size	(Un)Compressed Data Size (of single replica)	Ref
Beeline	Telecom	Data Platform	—	—	<a href="#">Blog   2021</a>
Benocs	Network Telemetry and Analytics	Main Product	—	—	<a href="#">Slides   Octok</a>
BIGO	Video	Computing Platform	—	—	<a href="#">Blog   Augu</a>
BiliBili	Video sharing	—	—	—	<a href="#">Blog   2021</a>
Bloomberg	Finance, Media	Monitoring	—	—	<a href="#">Job of Septe slides</a>
Bloxy	Blockchain	Analytics	—	—	<a href="#">Slides   Augu</a>
Bytedance	Social platforms	—	—	—	<a href="#">The C Meeti Octok</a>
CardsMobile	Finance	Analytics	—	—	<a href="#">VC.ru</a>
CARTO	Business Intelligence	Geo analytics	—	—	<a href="#">Geos proce ClickF</a>
CERN	Research	Experiment	—	—	<a href="#">Press 2012</a>
Checkly	Software Development	Analytics	—	—	<a href="#">Twee 2021</a>
ChelPipe Group	Analytics	—	—	—	<a href="#">Blog   2021</a>
Cisco	Networking	Traffic analysis	—	—	<a href="#">Lighti Octok</a>
Citadel Securities	Finance	—	—	—	<a href="#">Contr March</a>
Citymobil	Taxi	Analytics	—	—	<a href="#">Blog   Russi 2020</a>

Company	Industry	Usecase	Cluster Size	(Un)Compressed Data Size (of single replica)	Ref
Cloudflare	CDN	Traffic analysis	36 servers	—	Blog   2017 March
Comcast	Media	CDN Traffic Analysis	—	—	Apac Talk
ContentSquare	Web analytics	Main product	—	—	Blog   French 2018
Corunet	Analytics	Main product	—	—	Slides April
CreditX 氚信	Finance AI	Analysis	—	—	Slides Nov
Crazypanda	Games		—	—	Live session ClickMeet
Criteo	Retail	Main product	—	—	Slides Oct
Cryptology	Digital Assets Trading Platform	—	—	—	Job ad March
Dataliance for China Telecom	Telecom	Analytics	—	—	Slides Jan
Deutsche Bank	Finance	BI Analytics	—	—	Slides Oct
Deepplay	Gaming Analytics	—	—	—	Job ad 2020
Diva-e	Digital consulting	Main Product	—	—	Slides Sept
Ecommpay	Payment Processing	Logs	—	—	Video
Ecwid	E-commerce SaaS	Metrics, Logging	—	—	Slides April
eBay	E-commerce	Logs, Metrics and Events	—	—	Official Sep 2

Company	Industry	Usecase	Cluster Size	(Un)Compressed Data Size (of single replica)	Ref
Exness	Trading	Metrics, Logging	—	—	Talk i May 2
EventBunker.io	Serverless Data Processing	—	—	—	Tweet
FastNetMon	DDoS Protection	Main Product	—	—	Official
Flipkart	e-Commerce	—	—	—	Talk i July 2
FunCorp	Games	—	—	14 bn records/day as of Jan 2021	Article
Geniee	Ad network	Main product	—	—	Blog   Japan 2017
Genotek	Bioinformatics	Main product	—	—	Video 2020
Gigapipe	Managed ClickHouse	Main product	—	—	Official
Glaber	Monitoring	Main product	—	—	Webs
GraphCDN	CDN	Traffic Analytics	—	—	Blog   English 2021
HUYA	Video Streaming	Analytics	—	—	Slides Octo
Hydrolix	Cloud data platform	Main product	—	—	Docu
ICA	FinTech	Risk Management	—	—	Blog   English
Idealista	Real Estate	Analytics	—	—	Blog   English
Infobaleen	AI marketing tool	Analytics	—	—	Official
Infovista	Networks	Analytics	—	—	Slides Octo
InnoGames	Games	Metrics, Logging	—	—	Slides Septe

Company	Industry	Usecase	Cluster Size	(Un)Compressed Data Size (of single replica)	Ref
Instabug	APM Platform	Main product	—	—	A quick look at Instabug's log processing system
Instana	APM Platform	Main product	—	—	Twitter
Integros	Platform for video services	Analytics	—	—	Slides from May 2018
Ippon Technologies	Technology Consulting	—	—	—	Talk in July 2018
Ivi	Online Cinema	Analytics, Monitoring	—	—	Article Jan 2019
Jinshuju 金数据	BI Analytics	Main product	—	—	Slides from October 2018
kakaocorp	Internet company	—	—	—	if(kakao) conference
Kodiak Data	Clouds	Main product	—	—	Slides from April 2018
Kontur	Software Development	Metrics	—	—	Talk in November 2018
Kuaishou	Video	—	—	—	ClickHouse Meetup 2018
KGK Global	Vehicle monitoring	—	—	—	Press release 2021
Lawrence Berkeley National Laboratory	Research	Traffic analysis	1 server	11.8 TiB	Slides from April 2018
LifeStreet	Ad network	Main product	75 servers (3 replicas)	5.27 PiB	Blog post   Russian 2017
Mail.ru Cloud Solutions	Cloud services	Main product	—	—	Article
MAXILECT	Ad Tech, Blockchain, ML, AI	—	—	—	Job ad 2021

Company	Industry	Usecase	Cluster Size	(Un)Compressed Data Size (of single replica)	Refers to
Marilyn	Advertising	Statistics	—	—	Talk in June 2021
Mello	Marketing	Analytics	1 server	—	Article
MessageBird	Telecommunications	Statistics	—	—	Slides Nov 2020
Microsoft	Web Analytics	Clarity (Main Product)	—	—	A question GitHub
MindsDB	Machine Learning	Main Product	—	—	Official
MUX	Online Video	Video Analytics	—	—	Talk in August 2021
MGID	Ad network	Web-analytics	—	—	Blog   Russia 2020
Netskope	Network Security	—	—	—	Job ad March 2021
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 in July 2021
Ok.ru	Social Network	—	72 servers	810 TB compressed, 50bn rows/day, 1.5 TB/day	Smart conference 2021
Omnicomm	Transportation Monitoring	—	—	—	Facebook Oct 2021
OneAPM	Monitorings and Data Analysis	Main product	—	—	Slides Oct 2021
OZON	E-commerce	—	—	—	Official

Company	Industry	Usecase	Cluster Size	(Un)Compressed Data Size (of single replica)	Ref
Panelbear	Analytics	Monitoring and Analytics	—	—	Tech Note
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
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   2019
Raiffeisenbank	Banking	Analytics	—	—	Lecture Decem
Rambler	Internet services	Analytics	—	—	Talk i April
Replica	Urban Planning	Analytics	—	—	Job ad
Retell	Speech synthesis	Analytics	—	—	Blog , Augu
Rollbar	Software Development	Main Product	—	—	Official
Rspamd	Antispam	Analytics	—	—	Official
RuSIEM	SIEM	Main Product	—	—	Official

Company	Industry	Usecase	Cluster Size	(Un)Compressed Data Size (of single replica)	Refers to
S7 Airlines	Airlines	Metrics, Logging	—	—	Talk in March
Sber	Banking, Fintech, Retail, Cloud, Media	—	—	—	Job ad in March
scireum GmbH	e-Commerce	Main product	—	—	Talk in February
Segment	Data processing	Main product	9 * i3en.3xlarge nodes 7.5TB NVME SSDs, 96GB Memory, 12 vCPUs	—	Slides
sebot.io	Shopping Ads	—	—	—	A comparison Link
SEMrush	Marketing	Main product	—	—	Slides in August
Sentry	Software Development	Main product	—	—	Blog in English
seo.do	Analytics	Main product	—	—	Slides in November
SGK	Government Social Security	Analytics	—	—	Slides in November
SigNoz	Observability Platform	Main Product	—	—	Source code
Sina	News	—	—	—	Slides in October
Sipfront	Software Development	Analytics	—	—	Tweet in 2021
SMI2	News	Analytics	—	—	Blog in Russia in November
Spark New Zealand	Telecommunications	Security Operations	—	—	Blog in 2020

Company	Industry	Usecase	Cluster Size	(Un)Compressed Data Size (of single replica)	Ref
Splitbee	Analytics	Main Product	—	—	Blog   2021
Splunk	Business Analytics	Main product	—	—	Slides   Janua
Spotify	Music	Experimentation	—	—	Slides
Staffcop	Information Security	Main Product	—	—	Offici Docu
Suning	E-Commerce	User behaviour analytics	—	—	Blog
Teralytics	Mobility	Analytics	—	—	Tech
Tencent	Big Data	Data processing	—	—	Slides   Octob
Tencent	Messaging	Logging	—	—	Talk i Novem
Tencent Music Entertainment (TME)	BigData	Data processing	—	—	Blog   June 2
Tesla	Electric vehicle and clean energy company	—	—	—	Vacan descr 2021
Timeflow	Software	Analytics	—	—	Blog
Tinybird	Real-time Data Products	Data processing	—	—	Offici
Traffic Stars	AD network	—	300 servers in Europe/US	1.8 PiB, 700 000 insert rps (as of 2021)	Slides   May 2
Uber	Taxi	Logging	—	—	Slides   2020
UTMSTAT	Analytics	Main product	—	—	Blog   2020
VKontakte	Social Network	Statistics, Logging	—	—	Slides   Augus

Company	Industry	Usecase	Cluster Size	(Un)Compressed Data Size (of single replica)	Ref
VMware	Cloud	VeloCloud, SDN	—	—	Product doc
Walmart Labs	Internet, Retail	—	—	—	Talk i July 2
Wargaming	Games	—	—	—	Intern
Wildberries	E-commerce	—	—	—	Offici
Wisebits	IT Solutions	Analytics	—	—	Slides May 2
Workato	Automation Software	—	—	—	Talk i July 2
Xenoss	Marketing, Advertising	—	—	—	Instag 2021
Xiaoxin Tech	Education	Common purpose	—	—	Slides Nove
Ximalaya	Audio sharing	OLAP	—	—	Slides Nove
Yandex Cloud	Public Cloud	Main product	—	—	Talk i Decem
Yandex DataLens	Business Intelligence	Main product	—	—	Slides Decem
Yandex Market	e-Commerce	Metrics, Logging	—	—	Talk i Janua
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 2020
Yotascale	Cloud	Data pipeline	—	2 bn records/day	Link (Acco

Company	Industry	Usecase	Cluster Size	(Un)Compressed Data Size (of single replica)	Ref...
Zagrava Trading	—	—	—	—	Job of 2021
ЦВТ	Software Development	Metrics, Logging	—	—	Blog   2019
МКБ	Bank	Web-system monitoring	—	—	Slides   Septe...
ЦФТ	Banking, Financial products, Payments	—	—	—	Meeting   April
Цифровой Рабочий	Industrial IoT, Analytics	—	—	—	Blog   Russia 2021
ООО «МПЗ Богородский»	Agriculture	—	—	—	Article   Novem...
ДомКлик	Real Estate	—	—	—	Article   Octo...
Deepl	Machine Learning	—	—	—	Video   2021

## Начало работы

Если вы новичок в ClickHouse и хотите вживую оценить его производительность, прежде всего нужно пройти через [процесс установки](#).

После этого можно выбрать один из следующих вариантов:

- [Пройти подробное руководство для начинающих](#)
- [Поэкспериментировать с тестовыми наборами данных](#)

## Установка

### Системные требования

ClickHouse может работать на любой операционной системе Linux, FreeBSD или Mac OS X с архитектурой процессора x86\_64, AArch64 или PowerPC64LE.

Предварительно собранные пакеты компилируются для x86\_64 и используют набор инструкций SSE 4.2, поэтому, если не указано иное, его поддержка в используемом процессоре, становится дополнительным требованием к системе. Вот команда, чтобы проверить, поддерживает ли текущий процессор SSE 4.2:

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

Чтобы запустить ClickHouse на процессорах, которые не поддерживают SSE 4.2, либо имеют архитектуру AArch64 или PowerPC64LE, необходимо самостоятельно [собрать ClickHouse из исходного кода](#) с соответствующими настройками конфигурации.

## Доступные варианты установки

### Из DEB пакетов

Яндекс рекомендует использовать официальные скомпилированные deb пакеты для Debian или Ubuntu. Для установки пакетов выполните:

```
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
```

Также эти пакеты можно скачать и установить вручную отсюда:

<https://repo.clickhouse.com/deb/stable/main/>.

Чтобы использовать различные [версии ClickHouse](#) в зависимости от ваших потребностей, вы можете заменить stable на lts или testing.

Также вы можете вручную скачать и установить пакеты из [репозитория](#).

### Пакеты

- `clickhouse-common-static` — Устанавливает исполняемые файлы ClickHouse.
- `clickhouse-server` — Создает символические ссылки для `clickhouse-server` и устанавливает конфигурационные файлы.
- `clickhouse-client` — Создает символические ссылки для `clickhouse-client` и других клиентских инструментов и устанавливает конфигурационные файлы `clickhouse-client`.
- `clickhouse-common-static-dbg` — Устанавливает исполняемые файлы ClickHouse собранные с отладочной информацией.

## Внимание

Если вам нужно установить ClickHouse определенной версии, вы должны установить все пакеты одной версии:

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

### Из RPM пакетов

Команда ClickHouse в Яндексе рекомендует использовать официальные предкомпилированные rpm пакеты для CentOS, RedHat и всех остальных дистрибутивов Linux, основанных на rpm.

Сначала нужно подключить официальный репозиторий:

```
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
```

Для использования наиболее свежих версий нужно заменить `stable` на `testing` (рекомендуется для тестовых окружений). Также иногда доступен `prestable`.

Для, собственно, установки пакетов необходимо выполнить следующие команды:

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

Также есть возможность установить пакеты вручную, скачав отсюда:

[https://repo.clickhouse.com/rpm/stable/x86\\_64](https://repo.clickhouse.com/rpm/stable/x86_64).

## Из Tgz архивов

Команда ClickHouse в Яндексе рекомендует использовать предкомпилированные бинарники из `tgz` архивов для всех дистрибутивов, где невозможна установка `deb` и `rpm` пакетов.

Интересующую версию архивов можно скачать вручную с помощью `curl` или `wget` из репозитория <https://repo.clickhouse.com/tgz/>.

После этого архивы нужно распаковать и воспользоваться скриптами установки. Пример установки самой свежей версии:

```
export LATEST_VERSION=`curl https://api.github.com/repos/ClickHouse/ClickHouse/tags 2>/dev/null | grep -Eo '[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
```

Для production окружений рекомендуется использовать последнюю `stable`-версию. Её номер также можно найти на [github](https://github.com/ClickHouse/ClickHouse/tags) с на вкладке <https://github.com/ClickHouse/ClickHouse/tags> с постфиксом - `stable`.

## Из Docker образа

Для запуска ClickHouse в Docker нужно следовать инструкции на [Docker Hub](#). Внутри образов используются официальные `deb` пакеты.

## Из единого бинарного файла

Для установки ClickHouse под Linux можно использовать единый переносимый бинарный файл из последнего коммита ветки `master`: [<https://builds.clickhouse.com/master/amd64/clickhouse>].

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

## Из исполняемых файлов для нестандартных окружений

Для других операционных систем и архитектуры AArch64 сборки ClickHouse предоставляются в виде кросс-компилированного бинарного файла из последнего коммита ветки `master` (с задержкой в несколько часов).

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

После скачивания можно воспользоваться `clickhouse client` для подключения к серверу или `clickhouse local` для обработки локальных данных.

Чтобы установить ClickHouse в рамках всей системы (с необходимыми конфигурационными файлами, настройками пользователей и т.д.), выполните `sudo ./clickhouse install`. Затем выполните команды `clickhouse start` (чтобы запустить сервер) и `clickhouse-client` (чтобы подключиться к нему).

Данные сборки не рекомендуются для использования в рабочей среде, так как они недостаточно тщательно протестированы. Также в них присутствуют не все возможности ClickHouse.

## Из исходного кода

Для компиляции ClickHouse вручную, используйте инструкцию для [Linux](#) или [Mac OS X](#).

Можно скомпилировать пакеты и установить их, либо использовать программы без установки пакетов. Также при ручной сборке можно отключить необходимость поддержки набора инструкций SSE 4.2 или собрать под процессоры архитектуры AArch64.

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

Для работы собранного вручную сервера необходимо создать директории для данных и метаданных, а также сделать их `chown` для желаемого пользователя. Пути к этим директориям могут быть изменены в конфигурационном файле сервера (`src/programs/server/config.xml`), по умолчанию используются следующие:

```
/opt/clickhouse/data/default/  
/opt/clickhouse/metadata/default/
```

На Gentoo для установки ClickHouse из исходного кода можно использовать просто `emerge clickhouse`.

## Запуск

Для запуска сервера в качестве демона, выполните:

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

Смотрите логи в директории `/var/log/clickhouse-server/`.

Если сервер не стартует, проверьте корректность конфигурации в файле `/etc/clickhouse-server/config.xml`

Также можно запустить сервер вручную из консоли:

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

При этом, лог будет выводиться в консоль, что удобно для разработки.  
Если конфигурационный файл лежит в текущей директории, то указывать параметр `--config-file` не требуется, по умолчанию будет использован файл `./config.xml`.

После запуска сервера, соединиться с ним можно с помощью клиента командной строки:

```
clickhouse-client
```

По умолчанию он соединяется с `localhost:9000`, от имени пользователя `default` без пароля. Также клиент может быть использован для соединения с удалённым сервером с помощью аргумента `--host`.

Терминал должен использовать кодировку UTF-8.

Более подробная информация о клиенте располагается в разделе «[Клиент командной строки](#)».

Пример проверки работоспособности системы:

```
$ ./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 rows in set. Elapsed: 0.003 sec.

:)
```

## Поздравляем, система работает!

Для дальнейших экспериментов можно попробовать загрузить один из тестовых наборов данных или пройти [пошаговое руководство для начинающих](#).

# 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 doesn’t 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,
`SearchEnginID` 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),
`RemotelP` UInt32,
`RemotelP6` 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)

```

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,
```

```

`HasCLID` UInt8,
`FirstVisit` DateTime,
`PredLastVisit` Date,
`LastVisit` Date,
`TotalVisits` UInt32,
`TraficSource` Nested(
    ID Int8,
    SearchEnginID UInt16,
    AdvEnginID UInt8,
    PlaceID UInt16,
    SocialSourceNetworkID UInt8,
    Domain String,
    SearchPhrase String,
    SocialSourcePage String),
`Attendance` FixedString(16),
`CLID` UInt32,
`YCLID` UInt64,
`NormalizedRefererHash` UInt64,
`SearchPhraseHash` UInt64,
`RefererDomainHash` UInt64,
`NormalizedStartURLHash` UInt64,
`StartURLDomainHash` UInt64,
`NormalizedEndURLHash` UInt64,
`TopLevelDomain` UInt64,
`URLScheme` UInt64,
`OpenstatServiceNameHash` UInt64,
`OpenstatCampaignIDHash` UInt64,
`OpenstatAdIDHash` UInt64,
`OpenstatSourceIDHash` UInt64,
`UTMSourceHash` UInt64,
`UTMMediumHash` UInt64,
`UTMCampaignHash` UInt64,
`UTMContentHash` UInt64,
`UTMTermHash` UInt64,
`FromHash` UInt64,
`WebVisorEnabled` UInt8,
`WebVisorActivity` UInt32,
`ParsedParams` Nested(
    Key1 String,
    Key2 String,
    Key3 String,
    Key4 String,
    Key5 String,
    ValueDouble Float64),
`Market` Nested(
    Type UInt8,
    GoalID UInt32,
    OrderID String,
    OrderPrice Int64,
    PP UInt32,
    DirectPlaceID UInt32,
    DirectOrderID UInt32,
    DirectBannerID UInt32,
    GoodID String,
    GoodName String,
    GoodQuantity Int32,
    GoodPrice Int64),
`IslandID` FixedString(16)
)
ENGINE = CollapsingMergeTree(Sign)
PARTITION BY toYYYYMM(StartDate)
ORDER BY (CounterID, StartDate, intHash32(UserID), VisitID)
SAMPLE BY intHash32(UserID)

```

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](#) позволяет пользователям экспериментировать с ClickHouse, мгновенно выполняя запросы без настройки своего сервера или кластера.

В Playground доступны несколько тестовых массивов данных, а также примеры запросов, которые показывают возможности ClickHouse. Кроме того, вы можете выбрать LTS релиз ClickHouse, который хотите протестировать.

ClickHouse Playground дает возможность поработать с [Managed Service for ClickHouse](#) в конфигурации m2.small (4 vCPU, 32 ГБ ОЗУ), которую предоставляет [Яндекс.Облако](#). Дополнительную информацию об облачных провайдерах читайте в разделе [Поставщики облачных услуг ClickHouse](#).

Вы можете отправлять запросы к Playground с помощью любого HTTP-клиента, например [curl](#) или [wget](#), также можно установить соединение с помощью драйверов [JDBC](#) или [ODBC](#). Более подробная информация о программных продуктах, поддерживающих ClickHouse, доступна [здесь](#).

## Параметры доступа

Параметр	Значение
Конечная точка HTTPS	<a href="https://play-api.clickhouse.com:8443">https://play-api.clickhouse.com:8443</a>
Конечная точка TCP	play-api.clickhouse.com:9440

Параметр	Значение
Пользователь	playground
Пароль	clickhouse

Также можно подключаться к ClickHouse определённых релизов, чтобы протестировать их различия (порты и пользователь / пароль остаются неизменными):

- 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)

## Примечание

Для всех этих конечных точек требуется безопасное соединение TLS.

## Ограничения

Запросы выполняются под пользователем с правами `readonly`, для которого есть следующие ограничения:

- запрещены DDL запросы
- запрещены INSERT запросы

Также установлены следующие опции:

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

## Примеры

Пример конечной точки HTTPS с curl:

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

Пример конечной точки TCP с CLI:

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

## Детали реализации

Веб-интерфейс ClickHouse Playground выполняет запросы через ClickHouse [HTTP API](#).

Бэкэнд Playground - это кластер ClickHouse без дополнительных серверных приложений. Как упоминалось выше, способы подключения по HTTPS и TCP/TLS общедоступны как часть Playground. Они проксируются через [Cloudflare Spectrum](#) для добавления дополнительного уровня защиты и улучшенного глобального подключения.

## Предупреждение

Открывать сервер ClickHouse для публичного доступа в любой другой ситуации **настоятельно не рекомендуется**. Убедитесь, что он настроен только на частную сеть и защищен брандмауэром.

## 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](#).

## Тестовые массивы данных

Этот раздел описывает как получить тестовые массивы данных и загрузить их в ClickHouse. Для некоторых тестовых массивов данных также доступны тестовые запросы.

- [Анонимизированные данные Яндекс.Метрики](#)
- [Star Schema Benchmark](#)
- [Набор данных кулинарных рецептов](#)
- [WikiStat](#)
- [Терабайт логов кликов от Criteo](#)
- [AMPLab Big Data Benchmark](#)
- [Данные о такси в Нью-Йорке](#)
- [Набор данных о воздушном движении OpenSky Network 2020](#)
- [Данные о стоимости недвижимости в Великобритании](#)
- [OnTime](#)
- [Вышки сотовой связи](#)

## Анонимизированные данные Яндекс.Метрики

Датасет состоит из двух таблиц, содержащих анонимизированные данные о хитах (`hits_v1`) и визитах (`visits_v1`) Яндекс.Метрики. Каждую из таблиц можно скачать в виде сжатого `.tsv.xz`-файла или в виде уже готовых партиций. Также можно скачать расширенную версию таблицы `hits`, содержащую 100 миллионов строк в виде [архива с файлами TSV](#) и в виде [готовых партиций](#).

## Получение таблиц из партиций

### Скачивание и импортование партиций `hits`:

```
curl -O https://datasets.clickhouse.com/hits/partitions/hits_v1.tar  
tar xvf hits_v1.tar -C /var/lib/clickhouse # путь к папке с данными ClickHouse  
## убедитесь, что установлены корректные права доступа на файлы  
sudo service clickhouse-server restart  
clickhouse-client --query "SELECT COUNT(*) FROM datasets.hits_v1"
```

### Скачивание и импортование партиций `visits`:

```
curl -O https://datasets.clickhouse.com/visits/partitions/visits_v1.tar  
tar xvf visits_v1.tar -C /var/lib/clickhouse # путь к папке с данными ClickHouse  
## убедитесь, что установлены корректные права доступа на файлы  
sudo service clickhouse-server restart  
clickhouse-client --query "SELECT COUNT(*) FROM datasets.visits_v1"
```

## Получение таблиц из сжатых tsv-файлов

### **Скачивание и импорт в ClickHouse**

```

curl https://datasets.clickhouse.com/hits/tsv/hits_v1.tsv.xz | unxz --threads=`nproc` > hits_v1.tsv
## создадим таблицу hits_v1
clickhouse-client --query "CREATE DATABASE IF NOT EXISTS datasets"
clickhouse-client --query "CREATE TABLE datasets.hits_v1 ( WatchID UInt64, JavaEnable UInt8, Title String,
GoodEvent Int16, EventTime DateTime, EventDate Date, CounterID UInt32, ClientIP UInt32, ClientIP6
FixedString(16), RegionID UInt32, UserID UInt64, CounterClass 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 UInt8, SearchEngineID UInt16, SearchPhrase String, AdvEngineID UInt8,
IsArtifical UInt8, WindowClientWidth UInt16, WindowClientHeight UInt16, ClientTimeZone Int16, ClientEventTime
DateTime, SilverlightVersion1 UInt8, SilverlightVersion2 UInt8, SilverlightVersion3 UInt32, SilverlightVersion4
UInt16, PageCharset String, CodeVersion UInt32, IsLink UInt8, IsDownload UInt8, IsNotBounce UInt8, FUniqID
UInt64, HID UInt32, IsOldCounter UInt8, IsEvent UInt8, IsParameter UInt8, DontCountHits UInt8, WithHash UInt8,
HitColor FixedString(1), UTCEventTime DateTime, Age UInt8, Sex UInt8, Income UInt8, Interests UInt16, Robotness
UInt8, GeneralInterests Array(UInt16), RemoteIP UInt32, RemoteIP6 FixedString(16), WindowName Int32,
OpenerName Int32, HistoryLength Int16, BrowserLanguage FixedString(2), BrowserCountry FixedString(2),
SocialNetwork String, SocialAction String, HTTPError UInt16, SendTiming Int32, DNSTiming Int32, ConnectTiming
Int32, ResponseStartTiming Int32, ResponseEndTiming Int32, FetchTiming Int32, RedirectTiming Int32,
DOMInteractiveTiming Int32, DOMContentLoadedTiming Int32, DOMCompleteTiming Int32, LoadEventStartTiming
Int32, LoadEventEndTiming Int32, NSToDOMContentLoadedTiming Int32, FirstPaintTiming Int32, RedirectCount Int8,
SocialSourceNetworkID UInt8, SocialSourcePage String, ParamPrice Int64, ParamOrderID String, ParamCurrency
FixedString(3), ParamCurrencyID UInt16, GoalsReached Array(UInt32), OpenstatServiceName String,
OpenstatCampaignID String, OpenstatAdID String, OpenstatSourceID String, UTMSource String, UTMMedium String,
UTMCampaign String, UTMContent String, UTMTerm String, FromTag String, HasGCLID UInt8, RefererHash UInt64,
URLHash UInt64, CLID UInt32, YCLID UInt64, ShareService String, ShareURL String, ShareTitle String,
ParsedParams Nested(Key1 String, Key2 String, Key3 String, Key4 String, Key5 String, ValueDouble Float64),
IslandID FixedString(16), RequestNum UInt32, RequestTry UInt8) ENGINE = MergeTree() PARTITION BY
toYYYYMM(EventDate) ORDER BY (CounterID, EventDate, intHash32(userID)) SAMPLE BY intHash32(userID)
SETTINGS index_granularity = 8192"
## создадим таблицу 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 UInt8, SearchEngineID UInt16, SearchPhrase String, AdvEngineld 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"

## импортируем данные
cat hits_v1.tsv | clickhouse-client --query "INSERT INTO datasets.hits_v1 FORMAT TSV" --
max_insert_block_size=100000
## дополнительно можно оптимизировать таблицу
clickhouse-client --query "OPTIMIZE TABLE datasets.hits_v1 FINAL"
clickhouse-client --query "SELECT COUNT(*) FROM datasets.hits_v1"

```

## Скачивание и импортование visits из сжатого tsv-файла

```

curl https://datasets.clickhouse.com/visits/tsv/visits_v1.tsv.xz | unxz --threads=`nproc` > visits_v1.tsv
## теперь создадим таблицу
clickhouse-client --query "CREATE DATABASE IF NOT EXISTS datasets"
clickhouse-client --query "CREATE TABLE datasets.visits_v1 ( CounterID UInt32, StartDate Date, Sign Int8, IsNew
UInt8, VisitID UInt64, UserID UInt64, StartTime DateTime, Duration UInt32, UTCStartTime DateTime, PageViews
Int32, Hits Int32, IsBounce UInt8, Referer String, StartURL String, RefererDomain String, StartURLDomain String,
EndURL String, LinkURL String, IsDownload UInt8, TraficSourceID 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, BrowerCountry 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, AdvEnginID UInt8, PlaceID
UInt16, SocialSourceNetworkID UInt8, Domain String, SearchPhrase String, SocialSourcePage String), Attendance
FixedString(16), CLID UInt32, YCLID UInt64, NormalizedRefererHash UInt64, SearchPhraseHash UInt64,
RefererDomainHash UInt64, NormalizedStartURLHash UInt64, StartURLDomainHash UInt64, NormalizedEndURLHash
UInt64, TopLevelDomain UInt64, URLscheme UInt64, OpenstatServiceNameHash UInt64, OpenstatCampaignIDHash
UInt64, OpenstatAdIDHash UInt64, OpenstatSourceIDHash UInt64, UTMSourceHash UInt64, UTMMediumHash
UInt64, UTMCampaignHash UInt64, UTMContentHash UInt64, UTMTermHash UInt64, FromHash UInt64,
WebVisorEnabled UInt8, WebVisorActivity UInt32, ParsedParams Nested(Key1 String, Key2 String, Key3 String,
Key4 String, Key5 String, ValueDouble Float64), Market Nested(Type UInt8, GoalID UInt32, OrderID String,
OrderPrice Int64, PP UInt32, DirectPlaceID UInt32, DirectOrderID UInt32, DirectBannerID UInt32, GoodID String,
GoodName String, GoodQuantity Int32, GoodPrice Int64), IslandID FixedString(16)) ENGINE =
CollapsingMergeTree(Sign) PARTITION BY toYYYYMM(StartDate) ORDER BY (CounterID, StartDate, intHash32(UserID),
VisitID) SAMPLE BY intHash32(UserID) SETTINGS index_granularity = 8192"
## импортируем данные
cat visits_v1.tsv | clickhouse-client --query "INSERT INTO datasets.visits_v1 FORMAT TSV" --
max_insert_block_size=100000
## дополнительно можно оптимизировать таблицу
clickhouse-client --query "OPTIMIZE TABLE datasets.visits_v1 FINAL"
clickhouse-client --query "SELECT COUNT(*) FROM datasets.visits_v1"

```

## Запросы

Примеры запросов к этим таблицам (они называются `test.hits` и `test.visits`) можно найти среди **stateful тестов** и в некоторых **performance тестах** ClickHouse.

## Star Schema Benchmark

Компиляция dbgen:

```

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

```

Генерация данных:

### Внимание

-s 100 – dbgen генерирует 600 миллионов строк (67 ГБ)

-s 1000 – dbgen генерирует 6 миллиардов строк (занимает много времени)

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

Создание таблиц в Кликхауз:

```

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;

```

Вставка данных:

```

$ 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

```

Конвертация схемы-звезды в денормализованную плоскую схему:

```

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;
```

## Набор данных кулинарных рецептов

Набор данных кулинарных рецептов от RecipeNLG доступен для загрузки [здесь](#). Он содержит 2.2 миллиона рецептов, а его размер чуть меньше 1 ГБ.

### Загрузите и распакуйте набор данных

- Перейдите на страницу загрузки <https://recipenlg.cs.put.poznan.pl/dataset>.
- Примите Правила и условия и скачайте zip-архив с набором данных.
- Распакуйте zip-архив и вы получите файл full\_dataset.csv.

### Создайте таблицу

Запустите клиент ClickHouse и выполните следующий запрос для создания таблицы `recipes`:

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

## Добавьте данные в таблицу

Чтобы добавить данные из файла `full_dataset.csv` в таблицу `recipes`, выполните команду:

```
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"
```

Это один из примеров анализа пользовательских CSV-файлов с применением специальных настроек.

Пояснение:

- набор данных представлен в формате CSV и требует некоторой предварительной обработки при вставке. Для предварительной обработки используется табличная функция `input`;
- структура CSV-файла задается в аргументе табличной функции `input`;
- поле `num` (номер строки) не нужно — оно считывается из файла, но игнорируется;
- при загрузке используется `FORMAT CSVWithNames`, но заголовок в CSV будет проигнорирован (параметром командной строки `--input_format_with_names_use_header 0`), поскольку заголовок не содержит имени первого поля;
- в файле CSV для разделения строк используются только двойные кавычки. Но некоторые строки не заключены в двойные кавычки, и чтобы одинарная кавычка не рассматривалась как заключающая, используется параметр `--format_csv_allow_single_quote 0`;
- некоторые строки из CSV не могут быть считаны корректно, поскольку они начинаются с символов \M, тогда как в CSV начинаться с обратной косой черты могут только символы \N, которые распознаются как `NULL` в SQL. Поэтому используется параметр `--input_format_allow_errors_num 10`, разрешающий пропустить до десяти некорректных записей;
- массивы `ingredients`, `directions` и `NER` представлены в необычном виде: они сериализуются в строку формата JSON, а затем помещаются в CSV — тогда они могут считываться и обрабатываться как обычные строки (`String`). Чтобы преобразовать строку в массив, используется функция `JSONExtract`.

## Проверьте добавленные данные

Чтобы проверить добавленные данные, подсчитайте количество строк в таблице:

Запрос:

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

Результат:

```
count()  
2231141
```

## Примеры запросов

### Самые упоминаемые ингредиенты в рецептах:

В этом примере вы узнаете, как развернуть массив в набор строк с помощью функции `arrayJoin`.

Запрос:

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

Результат:

	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.)

## Самые сложные рецепты с клубникой

Запрос:

```

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

```

Результат:

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.)

В этом примере используется функция `has` для проверки вхождения элемента в массив, а также сортировка по количеству шагов (`length(directions)`).

Существует свадебный торт, который требует целых 126 шагов для производства! Рассмотрим эти шаги:

Запрос:

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

Результат:

```
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.  
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

Этот набор данных доступен в [Online Playground](#).

## WikiStat

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

Создание таблицы:

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

Загрузка данных:

```
$ 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
```

# Терабайт логов кликов от Criteo

Скачайте данные с <http://labs.criteo.com/downloads/download-terabyte-click-logs/>

Создайте таблицу для импорта лога:

```
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
```

Загрузите данные:

```
$ 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
```

Создайте таблицу для сконвертированных данных:

```
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)
```

Преобразуем данные из сырого лога и положим во вторую таблицу:

```
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

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

Зарегистрируйте бесплатную учетную запись на <https://aws.amazon.com> - понадобится кредитная карта, email и номер телефона

Получите новый ключ доступа на [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)

Выполните следующее в консоли:

```
$ 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 ..
```

Выполните следующие запросы к ClickHouse:

```
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);
```

Возвращаемся в консоль:

```
$ 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
```

Запросы для получения выборок данных:

```
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 — это аналитический тест производительности для данных журнала событий, сгенерированных машиной. Бенчмарк разработан [Andrew Crotty](#).

Скачать данные:

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

Распаковать данные:

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

Создание таблиц:

```

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);

```

Вставка данных:

```

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

```

Запуск тестов производительности:

-- 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
) 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;
```

Данные также доступны для работы с интерактивными запросами через [Playground](#), [пример](#).

## Данные о такси в Нью-Йорке

Этот датасет может быть получен двумя способами:

- импорт из сырых данных;
- скачивание готовых партиций.

## Как импортировать сырые данные

См. <https://github.com/toddwschneider/nyc-taxi-data> и <http://tech.marksblogg.com/billion-nyc-taxi-rides-redshift.html> для описания набора данных и инструкций по загрузке.

После скачивания получится порядка 227 Гб несжатых данных в CSV файлах. Скачивание занимает порядка часа на 1 Гбит соединении (параллельное скачивание с s3.amazonaws.com утилизирует как минимум половину гигабитного канала).

Некоторые файлы могут скачаться не полностью. Проверьте размеры файлов и скачайте повторно подозрительные.

Некоторые файлы могут содержать некорректные строки. Их можно скорректировать следующим образом:

```
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
```

Далее данные должны быть предобработаны в PostgreSQL. Будут сделаны выборки точек в полигонах (для установки соответствия точек на карте с районами Нью-Йорка) и объединение всех данных в одну денормализованную плоскую таблицу с помощью JOIN. Для этого потребуется установить PostgreSQL с поддержкой PostGIS.

При запуске `initialize_database.sh`, будьте осторожны и вручную перепроверьте, что все таблицы корректно создались.

Обработка каждого месяца данных в PostgreSQL занимает около 20-30 минут, в сумме порядка 48 часов.

Проверить количество загруженных строк можно следующим образом:

```
$ 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)

Данные в PostgreSQL занимают 370 Гб.

Экспорт данных из 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';
```

Слепок данных создается со скоростью около 50 Мб в секунду. Во время создания слепка, PostgreSQL читает с диска со скоростью около 28 Мб в секунду.  
Это занимает около 5 часов. Результирующий tsv файл имеет размер в 590612904969 байт.

Создание временной таблицы в 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_boroct2010 Nullable(String),
    pickup_cdeligibil Nullable(FixedString(1)),
    pickup_ntacode   Nullable(String),
    pickup_ntaname   Nullable(String),
    pickup_puma      Nullable(String),
    dropoff_nyct2010_gid Nullable(UInt8),
    dropoff_ctlabel  Nullable(String),
    dropoff_borocode Nullable(UInt8),
    dropoff_boroname Nullable(String),
    dropoff_ct2010    Nullable(String),
    dropoff_boroct2010 Nullable(String),
    dropoff_cdeligibil Nullable(String),
    dropoff_ntacode   Nullable(String),
    dropoff_ntaname   Nullable(String),
    dropoff_puma      Nullable(String)
) ENGINE = Log;
```

Она нужна для преобразование полей к более правильным типам данных и, если возможно, чтобы избавиться от NULL'ов.

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

Данные читаются со скоростью 112-140 Мб в секунду.

Загрузка данных в таблицу типа Log в один поток заняла 76 минут.

Данные в этой таблице занимают 142 Гб.

(Импорт данных напрямую из Postgres также возможен с использованием COPY ... TO PROGRAM.)

К сожалению, все поля, связанные с погодой (precipitation...average\_wind\_speed) заполнены NULL. Из-за этого мы исключим их из финального набора данных.

Для начала мы создадим таблицу на одном сервере. Позже мы сделаем таблицу распределенной.

Создадим и заполним итоговую таблицу:

```
CREATE TABLE trips_mergetree
ENGINE = MergeTree(pickup_date, pickup_datetime, 8192)
AS SELECT

trip_id,
CAST(vendor_id AS Enum8('1' = 1, '2' = 2, 'CMT' = 3, 'VTS' = 4, 'DDS' = 5, 'B02512' = 10, 'B02598' = 11, 'B02617' = 12, 'B02682' = 13, 'B02764' = 14)) AS vendor_id,
toDate(pickup_datetime) AS pickup_date,
ifNull(pickup_datetime, toDateTime(0)) AS pickup_datetime,
toDate(dropoff_datetime) AS dropoff_date,
ifNull(dropoff_datetime, toDateTime(0)) AS dropoff_datetime,
assumeNotNull(store_and_fwd_flag) IN ('Y', '1', '2') AS store_and_fwd_flag,
assumeNotNull(rate_code_id) AS rate_code_id,
assumeNotNull(pickup_longitude) AS pickup_longitude,
assumeNotNull(pickup_latitude) AS pickup_latitude,
assumeNotNull(dropoff_longitude) AS dropoff_longitude,
assumeNotNull(dropoff_latitude) AS dropoff_latitude,
assumeNotNull(passenger_count) AS passenger_count,
assumeNotNull(trip_distance) AS trip_distance,
assumeNotNull(fare_amount) AS fare_amount,
assumeNotNull(extra) AS extra,
assumeNotNull(mta_tax) AS mta_tax,
assumeNotNull(tip_amount) AS tip_amount,
assumeNotNull(tolls_amount) AS tolls_amount,
assumeNotNull(ehail_fee) AS ehail_fee,
assumeNotNull(improvement_surcharge) AS improvement_surcharge,
assumeNotNull(total_amount) AS total_amount,
CAST((assumeNotNull(payment_type) AS pt) IN ('CSH', 'CASH', 'Cash', 'CAS', 'Cas', '1') ? 'CSH' : (pt IN ('CRD', 'Credit', 'Cre', 'CRE', 'CREDIT', '2') ? 'CRE' : (pt IN ('NOC', 'No Charge', 'No', '3') ? 'NOC' : (pt IN ('DIS', 'Dispute', 'Dis', '4') ? 'DIS' : 'UNK'))) AS Enum8('CSH' = 1, 'CRE' = 2, 'UNK' = 0, 'NOC' = 3, 'DIS' = 4)) AS payment_type_,
assumeNotNull(trip_type) AS trip_type,
ifNull(toFixedString(unhex(pickup), 25), toFixedString("", 25)) AS pickup,
ifNull(toFixedString(unhex(dropoff), 25), toFixedString("", 25)) AS dropoff,
CAST(assumeNotNull(cab_type) AS Enum8('yellow' = 1, 'green' = 2, 'uber' = 3)) AS cab_type,

assumeNotNull(pickup_nyct2010_gid) AS pickup_nyct2010_gid,
toFloat32(ifNull(pickup_ctlabel, '0')) AS pickup_ctlabel,
assumeNotNull(pickup_borocode) AS pickup_borocode,
CAST(assumeNotNull(pickup_boroname) AS Enum8('Manhattan' = 1, 'Queens' = 4, 'Brooklyn' = 3, '' = 0, 'Bronx' = 2, 'Staten Island' = 5)) AS pickup_boroname,
toFixedString(ifNull(pickup_ct2010, '000000'), 6) AS pickup_ct2010,
toFixedString(ifNull(pickup_boroct2010, '0000000'), 7) AS pickup_boroct2010,
CAST(assumeNotNull(ifNull(pickup_cdeligibil, ' ')) AS Enum8(' ' = 0, 'E' = 1, 'I' = 2)) AS pickup_cdeligibil,
toFixedString(ifNull(pickup_ntacode, '0000'), 4) AS pickup_ntacode,

CAST(assumeNotNull(pickup_ntaname) AS Enum16(' ' = 0, 'Airport' = 1, 'Allerton-Pelham Gardens' = 2, 'Annadale-Huguenot-Prince's Bay-Eltingville' = 3, 'Arden Heights' = 4, 'Astoria' = 5, 'Auburndale' = 6, 'Baisley Park' = 7, 'Bath Beach' = 8, 'Battery Park City-Lower Manhattan' = 9, 'Bay Ridge' = 10, 'Bayside-Bayside Hills' = 11, 'Bedford' = 12, 'Bedford Park-Fordham North' = 13, 'Bellerose' = 14, 'Belmont' = 15, 'Bensonhurst East' = 16, 'Bensonhurst West' = 17, 'Borough Park' = 18, 'Breezy Point-Belle Harbor-Rockaway Park-Broad Channel' = 19, 'Briarwood-Jamaica Hills' = 20, 'Brighton Beach' = 21, 'Bronxdale' = 22, 'Brooklyn Heights-Cobble Hill' = 23, 'Brownsville' = 24, 'Bushwick North' = 25, 'Bushwick South' = 26, 'Cambria Heights' = 27, 'Canarsie' = 28, 'Carroll Gardens-Columbia Street-Red Hook' = 29, 'Central Harlem North-Polo Grounds' = 30, 'Central Harlem South' = 31, 'Charleston-Richmond Valley-Tottenville' = 32, 'Chinatown' = 33, 'Clarendon-Bathgate' = 34, 'Clinton' = 35, 'Clinton Hill' = 36, 'Co-op City' = 37, 'College Point' = 38, 'Corona' = 39, 'Crotona Park East' = 40, 'Crown Heights North' = 41, 'Crown Heights South' = 42, 'Cypress Hills-City Line' = 43, 'DUMBO-Vinegar Hill-Downtown Brooklyn-Boerum Hill' = 44, 'Douglas Manor-Douglas-Long Island 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' = 62, 'Gowanus' = 63, 'Greenpoint' = 64, 'Kingsbridge' = 65, 'Lefferts Gardens' = 66, 'Midwood' = 67, 'New Utrecht' = 68, 'Ozone Park' = 69, 'Prospect Heights' = 70, 'Red Hook' = 71, 'Sheepshead Bay' = 72, 'St. George' = 73, 'Tompkinsville' = 74, 'Washington Heights' = 75, 'West Farms' = 76, 'West New York' = 77, 'West Brighton' = 78, 'West End' = 79, 'West Farms' = 80, 'West Farms-Corona' = 81, 'West Side' = 82, 'West Village' = 83, 'Williamsburg' = 84, 'Woodhaven' = 85, 'Woodlawn' = 86, 'Woodlawn-Bronx' = 87, 'Woodlawn-University Heights' = 88, 'Woodlawn-West Farms' = 89, 'Woodlawn-West Farms-Bronx' = 90, 'Woodlawn-West Farms-Bronx' = 91, 'Woodlawn-West Farms-Bronx' = 92, 'Woodlawn-West Farms-Bronx' = 93, 'Woodlawn-West Farms-Bronx' = 94, 'Woodlawn-West Farms-Bronx' = 95, 'Woodlawn-West Farms-Bronx' = 96, 'Woodlawn-West Farms-Bronx' = 97, 'Woodlawn-West Farms-Bronx' = 98, 'Woodlawn-West Farms-Bronx' = 99, 'Woodlawn-West Farms-Bronx' = 100, 'Woodlawn-West Farms-Bronx' = 101, 'Woodlawn-West Farms-Bronx' = 102, 'Woodlawn-West Farms-Bronx' = 103, 'Woodlawn-West Farms-Bronx' = 104, 'Woodlawn-West Farms-Bronx' = 105, 'Woodlawn-West Farms-Bronx' = 106, 'Woodlawn-West Farms-Bronx' = 107, 'Woodlawn-West Farms-Bronx' = 108, 'Woodlawn-West Farms-Bronx' = 109, 'Woodlawn-West Farms-Bronx' = 110, 'Woodlawn-West Farms-Bronx' = 111, 'Woodlawn-West Farms-Bronx' = 112, 'Woodlawn-West Farms-Bronx' = 113, 'Woodlawn-West Farms-Bronx' = 114, 'Woodlawn-West Farms-Bronx' = 115, 'Woodlawn-West Farms-Bronx' = 116, 'Woodlawn-West Farms-Bronx' = 117, 'Woodlawn-West Farms-Bronx' = 118, 'Woodlawn-West Farms-Bronx' = 119, 'Woodlawn-West Farms-Bronx' = 120, 'Woodlawn-West Farms-Bronx' = 121, 'Woodlawn-West Farms-Bronx' = 122, 'Woodlawn-West Farms-Bronx' = 123, 'Woodlawn-West Farms-Bronx' = 124, 'Woodlawn-West Farms-Bronx' = 125, 'Woodlawn-West Farms-Bronx' = 126, 'Woodlawn-West Farms-Bronx' = 127, 'Woodlawn-West Farms-Bronx' = 128, 'Woodlawn-West Farms-Bronx' = 129, 'Woodlawn-West Farms-Bronx' = 130, 'Woodlawn-West Farms-Bronx' = 131, 'Woodlawn-West Farms-Bronx' = 132, 'Woodlawn-West Farms-Bronx' = 133, 'Woodlawn-West Farms-Bronx' = 134, 'Woodlawn-West Farms-Bronx' = 135, 'Woodlawn-West Farms-Bronx' = 136, 'Woodlawn-West Farms-Bronx' = 137, 'Woodlawn-West Farms-Bronx' = 138, 'Woodlawn-West Farms-Bronx' = 139, 'Woodlawn-West Farms-Bronx' = 140, 'Woodlawn-West Farms-Bronx' = 141, 'Woodlawn-West Farms-Bronx' = 142, 'Woodlawn-West Farms-Bronx' = 143, 'Woodlawn-West Farms-Bronx' = 144, 'Woodlawn-West Farms-Bronx' = 145, 'Woodlawn-West Farms-Bronx' = 146, 'Woodlawn-West Farms-Bronx' = 147, 'Woodlawn-West Farms-Bronx' = 148, 'Woodlawn-West Farms-Bronx' = 149, 'Woodlawn-West Farms-Bronx' = 150, 'Woodlawn-West Farms-Bronx' = 151, 'Woodlawn-West Farms-Bronx' = 152, 'Woodlawn-West Farms-Bronx' = 153, 'Woodlawn-West Farms-Bronx' = 154, 'Woodlawn-West Farms-Bronx' = 155, 'Woodlawn-West Farms-Bronx' = 156, 'Woodlawn-West Farms-Bronx' = 157, 'Woodlawn-West Farms-Bronx' = 158, 'Woodlawn-West Farms-Bronx' = 159, 'Woodlawn-West Farms-Bronx' = 160, 'Woodlawn-West Farms-Bronx' = 161, 'Woodlawn-West Farms-Bronx' = 162, 'Woodlawn-West Farms-Bronx' = 163, 'Woodlawn-West Farms-Bronx' = 164, 'Woodlawn-West Farms-Bronx' = 165, 'Woodlawn-West Farms-Bronx' = 166, 'Woodlawn-West Farms-Bronx' = 167, 'Woodlawn-West Farms-Bronx' = 168, 'Woodlawn-West Farms-Bronx' = 169, 'Woodlawn-West Farms-Bronx' = 170, 'Woodlawn-West Farms-Bronx' = 171, 'Woodlawn-West Farms-Bronx' = 172, 'Woodlawn-West Farms-Bronx' = 173, 'Woodlawn-West Farms-Bronx' = 174, 'Woodlawn-West Farms-Bronx' = 175, 'Woodlawn-West Farms-Bronx' = 176, 'Woodlawn-West Farms-Bronx' = 177, 'Woodlawn-West Farms-Bronx' = 178, 'Woodlawn-West Farms-Bronx' = 179, 'Woodlawn-West Farms-Bronx' = 180, 'Woodlawn-West Farms-Bronx' = 181, 'Woodlawn-West Farms-Bronx' = 182, 'Woodlawn-West Farms-Bronx' = 183, 'Woodlawn-West Farms-Bronx' = 184, 'Woodlawn-West Farms-Bronx' = 185, 'Woodlawn-West Farms-Bronx' = 186, 'Woodlawn-West Farms-Bronx' = 187, 'Woodlawn-West Farms-Bronx' = 188, 'Woodlawn-West Farms-Bronx' = 189, 'Woodlawn-West Farms-Bronx' = 190, 'Woodlawn-West Farms-Bronx' = 191, 'Woodlawn-West Farms-Bronx' = 192, 'Woodlawn-West Farms-Bronx' = 193, 'Woodlawn-West Farms-Bronx' = 194, 'Woodlawn-West Farms-Bronx' = 195, 'Woodlawn-West Farms-Bronx' = 196, 'Woodlawn-West Farms-Bronx' = 197, 'Woodlawn-West Farms-Bronx' = 198, 'Woodlawn-West Farms-Bronx' = 199, 'Woodlawn-West Farms-Bronx' = 200, 'Woodlawn-West Farms-Bronx' = 201, 'Woodlawn-West Farms-Bronx' = 202, 'Woodlawn-West Farms-Bronx' = 203, 'Woodlawn-West Farms-Bronx' = 204, 'Woodlawn-West Farms-Bronx' = 205, 'Woodlawn-West Farms-Bronx' = 206, 'Woodlawn-West Farms-Bronx' = 207, 'Woodlawn-West Farms-Bronx' = 208, 'Woodlawn-West Farms-Bronx' = 209, 'Woodlawn-West Farms-Bronx' = 210, 'Woodlawn-West Farms-Bronx' = 211, 'Woodlawn-West Farms-Bronx' = 212, 'Woodlawn-West Farms-Bronx' = 213, 'Woodlawn-West Farms-Bronx' = 214, 'Woodlawn-West Farms-Bronx' = 215, 'Woodlawn-West Farms-Bronx' = 216, 'Woodlawn-West Farms-Bronx' = 217, 'Woodlawn-West Farms-Bronx' = 218, 'Woodlawn-West Farms-Bronx' = 219, 'Woodlawn-West Farms-Bronx' = 220, 'Woodlawn-West Farms-Bronx' = 221, 'Woodlawn-West Farms-Bronx' = 222, 'Woodlawn-West Farms-Bronx' = 223, 'Woodlawn-West Farms-Bronx' = 224, 'Woodlawn-West Farms-Bronx' = 225, 'Woodlawn-West Farms-Bronx' = 226, 'Woodlawn-West Farms-Bronx' = 227, 'Woodlawn-West Farms-Bronx' = 228, 'Woodlawn-West Farms-Bronx' = 229, 'Woodlawn-West Farms-Bronx' = 230, 'Woodlawn-West Farms-Bronx' = 231, 'Woodlawn-West Farms-Bronx' = 232, 'Woodlawn-West Farms-Bronx' = 233, 'Woodlawn-West Farms-Bronx' = 234, 'Woodlawn-West Farms-Bronx' = 235, 'Woodlawn-West Farms-Bronx' = 236, 'Woodlawn-West Farms-Bronx' = 237, 'Woodlawn-West Farms-Bronx' = 238, 'Woodlawn-West Farms-Bronx' = 239, 'Woodlawn-West Farms-Bronx' = 240, 'Woodlawn-West Farms-Bronx' = 241, 'Woodlawn-West Farms-Bronx' = 242, 'Woodlawn-West Farms-Bronx' = 243, 'Woodlawn-West Farms-Bronx' = 244, 'Woodlawn-West Farms-Bronx' = 245, 'Woodlawn-West Farms-Bronx' = 246, 'Woodlawn-West Farms-Bronx' = 247, 'Woodlawn-West Farms-Bronx' = 248, 'Woodlawn-West Farms-Bronx' = 249, 'Woodlawn-West Farms-Bronx' = 250, 'Woodlawn-West Farms-Bronx' = 251, 'Woodlawn-West Farms-Bronx' = 252, 'Woodlawn-West Farms-Bronx' = 253, 'Woodlawn-West Farms-Bronx' = 254, 'Woodlawn-West Farms-Bronx' = 255, 'Woodlawn-West Farms-Bronx' = 256, 'Woodlawn-West Farms-Bronx' = 257, 'Woodlawn-West Farms-Bronx' = 258, 'Woodlawn-West Farms-Bronx' = 259, 'Woodlawn-West Farms-Bronx' = 260, 'Woodlawn-West Farms-Bronx' = 261, 'Woodlawn-West Farms-Bronx' = 262, 'Woodlawn-West Farms-Bronx' = 263, 'Woodlawn-West Farms-Bronx' = 264, 'Woodlawn-West Farms-Bronx' = 265, 'Woodlawn-West Farms-Bronx' = 266, 'Woodlawn-West Farms-Bronx' = 267, 'Woodlawn-West Farms-Bronx' = 268, 'Woodlawn-West Farms-Bronx' = 269, 'Woodlawn-West Farms-Bronx' = 270, 'Woodlawn-West Farms-Bronx' = 271, 'Woodlawn-West Farms-Bronx' = 272, 'Woodlawn-West Farms-Bronx' = 273, 'Woodlawn-West Farms-Bronx' = 274, 'Woodlawn-West Farms-Bronx' = 275, 'Woodlawn-West Farms-Bronx' = 276, 'Woodlawn-West Farms-Bronx' = 277, 'Woodlawn-West Farms-Bronx' = 278, 'Woodlawn-West Farms-Bronx' = 279, 'Woodlawn-West Farms-Bronx' = 280, 'Woodlawn-West Farms-Bronx' = 281, 'Woodlawn-West Farms-Bronx' = 282, 'Woodlawn-West Farms-Bronx' = 283, 'Woodlawn-West Farms-Bronx' = 284, 'Woodlawn-West Farms-Bronx' = 285, 'Woodlawn-West Farms-Bronx' = 286, 'Woodlawn-West Farms-Bronx' = 287, 'Woodlawn-West Farms-Bronx' = 288, 'Woodlawn-West Farms-Bronx' = 289, 'Woodlawn-West Farms-Bronx' = 290, 'Woodlawn-West Farms-Bronx' = 291, 'Woodlawn-West Farms-Bronx' = 292, 'Woodlawn-West Farms-Bronx' = 293, 'Woodlawn-West Farms-Bronx' = 294, 'Woodlawn-West Farms-Bronx' = 295, 'Woodlawn-West Farms-Bronx' = 296, 'Woodlawn-West Farms-Bronx' = 297, 'Woodlawn-West Farms-Bronx' = 298, 'Woodlawn-West Farms-Bronx' = 299, 'Woodlawn-West Farms-Bronx' = 300, 'Woodlawn-West Farms-Bronx' = 301, 'Woodlawn-West Farms-Bronx' = 302, 'Woodlawn-West Farms-Bronx' = 303, 'Woodlawn-West Farms-Bronx' = 304, 'Woodlawn-West Farms-Bronx' = 305, 'Woodlawn-West Farms-Bronx' = 306, 'Woodlawn-West Farms-Bronx' = 307, 'Woodlawn-West Farms-Bronx' = 308, 'Woodlawn-West Farms-Bronx' = 309, 'Woodlawn-West Farms-Bronx' = 310, 'Woodlawn-West Farms-Bronx' = 311, 'Woodlawn-West Farms-Bronx' = 312, 'Woodlawn-West Farms-Bronx' = 313, 'Woodlawn-West Farms-Bronx' = 314, 'Woodlawn-West Farms-Bronx' = 315, 'Woodlawn-West Farms-Bronx' = 316, 'Woodlawn-West Farms-Bronx' = 317, 'Woodlawn-West Farms-Bronx' = 318, 'Woodlawn-West Farms-Bronx' = 319, 'Woodlawn-West Farms-Bronx' = 320, 'Woodlawn-West Farms-Bronx' = 321, 'Woodlawn-West Farms-Bronx' = 322, 'Woodlawn-West Farms-Bronx' = 323, 'Woodlawn-West Farms-Bronx' = 324, 'Woodlawn-West Farms-Bronx' = 325, 'Woodlawn-West Farms-Bronx' = 326, 'Woodlawn-West Farms-Bronx' = 327, 'Woodlawn-West Farms-Bronx' = 328, 'Woodlawn-West Farms-Bronx' = 329, 'Woodlawn-West Farms-Bronx' = 330, 'Woodlawn-West Farms-Bronx' = 331, 'Woodlawn-West Farms-Bronx' = 332, 'Woodlawn-West Farms-Bronx' = 333, 'Woodlawn-West Farms-Bronx' = 334, 'Woodlawn-West Farms-Bronx' = 335, 'Woodlawn-West Farms-Bronx' = 336, 'Woodlawn-West Farms-Bronx' = 337, 'Woodlawn-West Farms-Bronx' = 338, 'Woodlawn-West Farms-Bronx' = 339, 'Woodlawn-West Farms-Bronx' = 340, 'Woodlawn-West Farms-Bronx' = 341, 'Woodlawn-West Farms-Bronx' = 342, 'Woodlawn-West Farms-Bronx' = 343, 'Woodlawn-West Farms-Bronx' = 344, 'Woodlawn-West Farms-Bronx' = 345, 'Woodlawn-West Farms-Bronx' = 346, 'Woodlawn-West Farms-Bronx' = 347, 'Woodlawn-West Farms-Bronx' = 348, 'Woodlawn-West Farms-Bronx' = 349, 'Woodlawn-West Farms-Bronx' = 350, 'Woodlawn-West Farms-Bronx' = 351, 'Woodlawn-West Farms-Bronx' = 352, 'Woodlawn-West Farms-Bronx' = 353, 'Woodlawn-West Farms-Bronx' = 354, 'Woodlawn-West Farms-Bronx' = 355, 'Woodlawn-West Farms-Bronx' = 356, 'Woodlawn-West Farms-Bronx' = 357, 'Woodlawn-West Farms-Bronx' = 358, 'Woodlawn-West Farms-Bronx' = 359, 'Woodlawn-West Farms-Bronx' = 360, 'Woodlawn-West Farms-Bronx' = 361, 'Woodlawn-West Farms-Bronx' = 362, 'Woodlawn-West Farms-Bronx' = 363, 'Woodlawn-West Farms-Bronx' = 364, 'Woodlawn-West Farms-Bronx' = 365, 'Woodlawn-West Farms-Bronx' = 366, 'Woodlawn-West Farms-Bronx' = 367, 'Woodlawn-West Farms-Bronx' = 368, 'Woodlawn-West Farms-Bronx' = 369, 'Woodlawn-West Farms-Bronx' = 370, 'Woodlawn-West Farms-Bronx' = 371, 'Woodlawn-West Farms-Bronx' = 372, 'Woodlawn-West Farms-Bronx' = 373, 'Woodlawn-West Farms-Bronx' = 374, 'Woodlawn-West Farms-Bronx' = 375, 'Woodlawn-West Farms-Bronx' = 376, 'Woodlawn-West Farms-Bronx' = 377, 'Woodlawn-West Farms-Bronx' = 378, 'Woodlawn-West Farms-Bronx' = 379, 'Woodlawn-West Farms-Bronx' = 380, 'Woodlawn-West Farms-Bronx' = 381, 'Woodlawn-West Farms-Bronx' = 382, 'Woodlawn-West Farms-Bronx' = 383, 'Woodlawn-West Farms-Bronx' = 384, 'Woodlawn-West Farms-Bronx' = 385, 'Woodlawn-West Farms-Bronx' = 386, 'Woodlawn-West Farms-Bronx' = 387, 'Woodlawn-West Farms-Bronx' = 388, 'Woodlawn-West Farms-Bronx' = 389, 'Woodlawn-West Farms-Bronx' = 390, 'Woodlawn-West Farms-Bronx' = 391, 'Woodlawn-West Farms-Bronx' = 392, 'Woodlawn-West Farms-Bronx' = 393, 'Woodlawn-West Farms-Bronx' = 394, 'Woodlawn-West Farms-Bronx' = 395, 'Woodlawn-West Farms-Bronx' = 396, 'Woodlawn-West Farms-Bronx' = 397, 'Woodlawn-West Farms-Bronx' = 398, 'Woodlawn-West Farms-Bronx' = 399, 'Woodlawn-West Farms-Bronx' = 400, 'Woodlawn-West Farms-Bronx' = 401, 'Woodlawn-West Farms-Bronx' = 402, 'Woodlawn-West Farms-Bronx' = 403, 'Woodlawn-West Farms-Bronx' = 404, 'Woodlawn-West Farms-Bronx' = 405, 'Woodlawn-West Farms-Bronx' = 406, 'Woodlawn-West Farms-Bronx' = 407, 'Woodlawn-West Farms-Bronx' = 408, 'Woodlawn-West Farms-Bronx' = 409, 'Woodlawn-West Farms-Bronx' = 410, 'Woodlawn-West Farms-Bronx' = 411, 'Woodlawn-West Farms-Bronx' = 412, 'Woodlawn-West Farms-Bronx' = 413, 'Woodlawn-West Farms-Bronx' = 414, 'Woodlawn-West Farms-Bronx' = 415, 'Woodlawn-West Farms-Bronx' = 416, 'Woodlawn-West Farms-Bronx' = 417, 'Woodlawn-West Farms-Bronx' = 418, 'Woodlawn-West Farms-Bronx' = 419, 'Woodlawn-West Farms-Bronx' = 420, 'Woodlawn-West Farms-Bronx' = 421, 'Woodlawn-West Farms-Bronx' = 422, 'Woodlawn-West Farms-Bronx' = 423, 'Woodlawn-West Farms-Bronx' = 424, 'Woodlawn-West Farms-Bronx' = 425, 'Woodlawn-West Farms-Bronx' = 426, 'Woodlawn-West Farms-Bronx' = 427, 'Woodlawn-West Farms-Bronx' = 428, 'Woodlawn-West Farms-Bronx' = 429, 'Woodlawn-West Farms-Bronx' = 430, 'Woodlawn-West Farms-Bronx' = 431, 'Woodlawn-West Farms-Bronx' = 432, 'Woodlawn-West Farms-Bronx' = 433, 'Woodlawn-West Farms-Bronx' = 434, 'Woodlawn-West Farms-Bronx' = 435, 'Woodlawn-West Farms-Bronx' = 436, 'Woodlawn-West Farms-Bronx' = 437, 'Woodlawn-West Farms-Bronx' = 438, 'Woodlawn-West Farms-Bronx' = 439, 'Woodlawn-West Farms-Bronx' = 440, 'Woodlawn-West Farms-Bronx' = 441, 'Woodlawn-West Farms-Bronx' = 442, 'Woodlawn-West Farms-Bronx' = 443, 'Woodlawn-West Farms-Bronx' = 444, 'Woodlawn-West Farms-Bronx' = 445, 'Woodlawn-West Farms-Bronx' = 446, 'Woodlawn-West Farms-Bronx' = 447, 'Woodlawn-West Farms-Bronx' = 448, 'Woodlawn-West Farms-Bronx' = 449, 'Woodlawn-West Farms-Bronx' = 450, 'Woodlawn-West Farms-Bronx' = 451, 'Woodlawn-West Farms-Bronx' = 452, 'Woodlawn-West Farms-Bronx' = 453, 'Woodlawn-West Farms-Bronx' = 454, 'Woodlawn-West Farms-Bronx' = 455, 'Woodlawn-West Farms-Bronx' = 456, 'Woodlawn-West Farms-Bronx' = 457, 'Woodlawn-West Farms-Bronx' = 458, 'Woodlawn-West Farms-Bronx' = 459, 'Woodlawn-West Farms-Bronx' = 460, 'Woodlawn-West Farms-Bronx' = 461, 'Woodlawn-West Farms-Bronx' = 462, 'Woodlawn-West Farms-Bronx' = 463, 'Woodlawn-West Farms-Bronx' = 464, 'Woodlawn-West Farms-Bronx' = 465, 'Woodlawn-West Farms-Bronx' = 466, 'Woodlawn-West Farms-Bronx' = 467, 'Woodlawn-West Farms-Bronx' = 468, 'Woodlawn-West Farms-Bronx' = 469, 'Woodlawn-West Farms-Bronx' = 470, 'Woodlawn-West Farms-Bronx' = 471, 'Woodlawn-West Farms-Bronx' = 472, 'Woodlawn-West Farms-Bronx' = 473, 'Woodlawn-West Farms-Bronx' = 474, 'Woodlawn-West Farms-Bronx' = 475, 'Woodlawn-West Farms-Bronx' = 476, 'Woodlawn-West Farms-Bronx' = 477, 'Woodlawn-West Farms-Bronx' = 478, 'Woodlawn-West Farms-Bronx' = 479, 'Woodlawn-West Farms-Bronx' = 480, 'Woodlawn-West Farms-Bronx' = 481, 'Woodlawn-West Farms-Bronx' = 482, 'Woodlawn-West Farms-Bronx' = 483, 'Woodlawn-West Farms-Bronx' = 484, 'Woodlawn-West Farms-Bronx' = 485, 'Woodlawn-West Farms-Bronx' = 486, 'Woodlawn-West Farms-Bronx' = 487, 'Woodlawn-West Farms-Bronx' = 488, 'Woodlawn-West Farms-Bronx' = 489, 'Woodlawn-West Farms-Bronx' = 490, 'Woodlawn-West Farms-Bronx' = 491, 'Woodlawn-West Farms-Bronx' = 492, 'Woodlawn-West Farms-Bronx' = 493, 'Woodlawn-West Farms-Bronx' = 494, 'Woodlawn-West Farms-Bronx' = 495, 'Woodlawn-West Farms-Bronx' = 496, 'Woodlawn-West Farms-Bronx' = 497, 'Woodlawn-West Farms-Bronx' = 498, 'Woodlawn-West Farms-Bronx' = 499, 'Woodlawn-West Farms-Bronx' = 500, 'Woodlawn-West Farms-Bronx' = 501, 'Woodlawn-West Farms-Bronx' = 502, 'Woodlawn-West Farms-Bronx' = 503, 'Woodlawn-West Farms-Bronx' = 504, 'Woodlawn-West Farms-Bronx' = 505, 'Woodlawn-West Farms-Bronx' = 506, 'Woodlawn-West Farms-Bronx' = 507, 'Woodlawn-West Farms-Bronx' = 508, 'Woodlawn-West Farms-Bronx' = 509, 'Woodlawn-West Farms-Bronx' = 510, 'Woodlawn-West Farms-Bronx' = 511, 'Woodlawn-West Farms-Bronx' = 512, 'Woodlawn-West Farms-Bronx' = 513, 'Woodlawn-West Farms-Bronx' = 514, 'Woodlawn-West Farms-Bronx' = 515, 'Woodlawn-West Farms-Bronx' = 516, 'Woodlawn-West Farms-Bronx' = 517, 'Woodlawn-West Farms-Bronx' = 518, 'Woodlawn-West Farms-Bronx' = 519, 'Woodlawn-West Farms-Bronx' = 520, 'Woodlawn-West Farms-Bronx' = 521, 'Woodlawn-West Farms-Bronx' = 522, 'Woodlawn-West Farms-Bronx' = 523, 'Woodlawn-West Farms-Bronx' = 524, 'Woodlawn-West Farms-Bronx' = 525, 'Woodlawn-West Farms-Bronx' = 526, 'Woodlawn-West Farms-Bronx' = 527, 'Woodlawn-West Farms-Bronx' = 528, 'Woodlawn-West Farms-Bronx' = 529, 'Woodlawn-West Farms-Bronx' = 530, 'Woodlawn-West Farms-Bronx' = 531, 'Woodlawn-West Farms-Bronx' = 532, 'Woodlawn-West Farms-Bronx' = 533, 'Woodlawn-West Farms-Bronx' = 534, 'Woodlawn-West Farms-Bronx' = 535, 'Woodlawn-West Farms-Bronx' = 536, 'Woodlawn-West Farms-Bronx' = 537, 'Woodlawn-West Farms-Bronx' = 538, 'Woodlawn-West Farms-Bronx' = 539, 'Woodlawn-West Farms-Bronx' = 540, 'Woodlawn-West Farms-Bronx' = 541, 'Woodlawn-West Farms-Bronx' = 542, 'Woodlawn-West Farms-Bronx' = 543, 'Woodlawn-West Farms-Bronx' = 544, 'Woodlawn-West Farms-Bronx' = 545, 'Woodlawn-West Farms-Bronx' = 546, 'Woodlawn-West Farms-Bronx' = 547, 'Woodlawn-West Farms-Bronx' = 548, 'Woodlawn-West Farms-Bronx' = 549, 'Woodlawn-West Farms-Bronx' = 550, 'Woodlawn-West Farms-Bronx' = 551, 'Woodlawn-West Farms-Bronx' = 552, 'Woodlawn-West Farms-Bronx' = 553, 'Woodlawn-West Farms-Bronx' = 554, 'Woodlawn-West Farms-Bronx' = 555, 'Woodlawn-West Farms-Bronx' = 556, 'Woodlawn-West Farms-Bronx' = 557, 'Woodlawn-West Farms-Bronx' = 558, 'Woodlawn-West Farms-Bronx' = 559, 'Woodlawn-West Farms-Bronx' = 560, 'Woodlawn-West Farms-Bronx' = 561, 'Woodlawn-West Farms-Bronx' = 562, 'Woodlawn-West Farms-Bronx' = 563, 'Woodlawn-West Farms-Bronx' = 564, 'Woodlawn-West Farms-Bronx' = 565, 'Woodlawn-West Farms-Bronx' = 566, 'Woodlawn-West Farms-Bronx' = 567, 'Woodlawn-West Farms-Bronx' = 568, 'Woodlawn-West Farms-Bronx' = 569, 'Woodlawn-West Farms-Bronx' = 570, 'Woodlawn-West Farms-Bronx' = 571, 'Woodlawn-West Farms-Bronx' = 572, 'Woodlawn-West Farms-Bronx' = 573, 'Woodlawn-West Farms-Bronx' = 574, 'Woodlawn-West Farms-Bronx' = 575, 'Woodlawn-West Farms-Bronx' = 576, 'Woodlawn-West Farms-Bronx' = 577, 'Woodlawn-West Farms-Bronx' = 578, 'Woodlawn-West Farms-Bronx' = 579, 'Woodlawn-West Farms-Bronx' = 580, 'Woodlawn-West Farms-Bronx' = 581, 'Woodlawn-West Farms-Bronx' = 582, 'Woodlawn-West Farms-Bronx' = 583, 'Woodlawn-West Farms-Bronx' = 584, 'Woodlawn-West Farms-Bronx' = 585, 'Woodlawn-West Farms-Bronx' = 586, 'Woodlawn-West Farms-Bronx' = 587, 'Woodlawn-West Farms-Bronx' = 588, 'Woodlawn-West Farms-Bronx' = 589, 'Woodlawn-West Farms-Bronx' = 590, 'Woodlawn-West Farms-Bronx' = 591, 'Woodlawn-West Farms-Bronx' = 592, 'Woodlawn-West Farms-Bronx' = 593, 'Woodlawn-West Farms-Bronx' = 594, 'Woodlawn-West Farms-Bronx' = 595, 'Woodlawn-West Farms-Bronx' = 596, 'Woodlawn-West Farms-Bronx' = 597, 'Woodlawn-West Farms-Bronx' = 598, 'Woodlawn-West Farms-Bronx' = 599, 'Woodlawn-West Farms-Bronx' = 600, 'Woodlawn-West Farms-Bronx' = 601, 'Woodlawn-West Farms-Bronx' = 602, 'Woodlawn-West Farms-Bronx' = 603, 'Woodlawn-West Farms-Bronx' = 604, 'Woodlawn-West Farms-Bronx' = 605, 'Woodlawn-West Farms-Bronx' = 606, 'Woodlawn-West Farms-Bronx' = 607, 'Woodlawn-West Farms-Bronx' = 608, 'Woodlawn-West Farms-Bronx' = 609, 'Woodlawn-West Farms-Bronx' = 610, 'Woodlawn-West Farms-Bronx' = 611, 'Woodlawn-West Farms-Bronx' = 612, 'Woodlawn-West Farms-Bronx' = 613, 'Woodlawn-West Farms-Bronx' = 614, 'Woodlawn-West Farms-Bronx' = 615, 'Woodlawn-West Farms-Bronx' = 616, 'Woodlawn-West Farms-Bronx' = 617, 'Woodlawn-West Farms-Bronx' = 618, 'Woodlawn-West Farms-Bronx' = 619, 'Woodlawn-West Farms-Bronx' = 620, 'Woodlawn-West Farms-Bronx' = 621, 'Woodlawn-West Farms-Bronx' = 622, 'Woodlawn-West Farms-Bronx' = 623, 'Woodlawn-West Farms-Bronx' = 624, 'Woodlawn-West Farms-Bronx' = 625, 'Woodlawn-West Farms-Bronx' = 626, 'Woodlawn-West Farms-Bronx' = 627, 'Woodlawn-West Farms-Bronx' = 628, 'Woodlawn-West Farms-Bronx' = 629, 'Woodlawn-West Farms-Bronx' = 630, 'Woodlawn-West Farms-Bronx' = 631, 'Woodlawn-West Farms-Bronx' = 632, 'Woodlawn-West Farms-Bronx' = 633, 'Woodlawn-West Farms-Bronx' = 634, 'Woodlawn-West Farms-Bronx' = 635, 'Woodlawn-West Farms-Bronx' = 636, 'Woodlawn-West Farms-Bronx' = 637, 'Woodlawn-West Farms-Bronx' = 638, 'Woodlawn-West Farms-Bronx' = 639, 'Woodlawn-West Farms-Bronx' = 640, 'Woodlawn-West Farms-Bronx' = 641, 'Woodlawn-West Farms-Bronx' = 642, 'Woodlawn-West Farms-Bronx' = 643, 'Woodlawn-West Farms-Bronx
```

Rockaway-Bayswater' = 62, 'Flatbush' = 63, 'Flatlands' = 64, 'Flushing' = 65, 'Fordham South' = 66, 'Forest Hills' = 67, 'Fort Greene' = 68, 'Fresh Meadows-Utopia' = 69, 'Ft. Totten-Bay Terrace-Clearview' = 70, 'Georgetown-Marine Park-Bergen Beach-Mill Basin' = 71, 'Glen Oaks-Floral Park-New Hyde Park' = 72, 'Glendale' = 73, 'Gramercy' = 74, 'Grasmere-Arrochar-Ft. Wadsworth' = 75, 'Gravesend' = 76, 'Great Kills' = 77, 'Greenpoint' = 78, 'Grymes Hill-Clifton-Fox Hills' = 79, 'Hamilton Heights' = 80, 'Hammels-Arverne-Edgemere' = 81, 'Highbridge' = 82, 'Hollis' = 83, 'Homecrest' = 84, 'Hudson Yards-Chelsea-Flatiron-Union Square' = 85, 'Hunters Point-Sunnyside-West Maspeth' = 86, 'Hunts Point' = 87, 'Jackson Heights' = 88, 'Jamaica' = 89, 'Jamaica Estates-Holliswood' = 90, 'Kensington-Ocean Parkway' = 91, 'Kew Gardens' = 92, 'Kew Gardens Hills' = 93, 'Kingsbridge Heights' = 94, 'Laurelton' = 95, 'Lenox Hill-Roosevelt Island' = 96, 'Lincoln Square' = 97, 'Lindenwood-Howard Beach' = 98, 'Longwood' = 99, 'Lower East Side' = 100, 'Madison' = 101, 'Manhattanville' = 102, 'Marble Hill-Inwood' = 103, 'Mariner's Harbor-Arlington-Port Ivory-Graniteville' = 104, 'Maspeth' = 105, 'Melrose South-Mott Haven North' = 106, 'Middle Village' = 107, 'Midtown-Midtown South' = 108, 'Midwood' = 109, 'Morningside Heights' = 110, 'Morrisania-Melrose' = 111, 'Mott Haven-Port Morris' = 112, 'Mount Hope' = 113, 'Murray Hill' = 114, 'Murray Hill-Kips Bay' = 115, 'New Brighton-Silver Lake' = 116, 'New Dorp-Midland Beach' = 117, 'New Springville-Bloomfield-Travis' = 118, 'North Corona' = 119, 'North Riverdale-Fieldston-Riverdale' = 120, 'North Side-South Side' = 121, 'Norwood' = 122, 'Oakland Gardens' = 123, 'Oakwood-Oakwood Beach' = 124, 'Ocean Hill' = 125, 'Ocean Parkway South' = 126, 'Old Astoria' = 127, 'Old Town-Dongan Hills-South Beach' = 128, 'Ozone Park' = 129, 'Park Slope-Gowanus' = 130, 'Parkchester' = 131, 'Pelham Bay-Country Club-City Island' = 132, 'Pelham Parkway' = 133, 'Pomonok-Flushing Heights-Hillcrest' = 134, 'Port Richmond' = 135, 'Prospect Heights' = 136, 'Prospect Lefferts Gardens-Wingate' = 137, 'Queens Village' = 138, 'Queensboro Hill' = 139, 'Queensbridge-Ravenswood-Long Island City' = 140, 'Rego Park' = 141, 'Richmond Hill' = 142, 'Ridgewood' = 143, 'Rikers Island' = 144, 'Rosedale' = 145, 'Rossville-Woodrow' = 146, 'Rugby-Remsen Village' = 147, 'Schuylerville-Throgs Neck-Edgewater Park' = 148, 'Seagate-Coney Island' = 149, 'Sheepshead Bay-Gerritsen Beach-Manhattan Beach' = 150, 'SoHo-TriBeCa-Civic Center-Little Italy' = 151, 'Soundview-Bruckner' = 152, 'Soundview-Castle Hill-Clason Point-Harding Park' = 153, 'South Jamaica' = 154, 'South Ozone Park' = 155, 'Springfield Gardens North' = 156, 'Springfield Gardens South-Brookville' = 157, 'Spuyten Duyvil-Kingsbridge' = 158, 'St. Albans' = 159, 'Stapleton-Rosebank' = 160, 'Starrett City' = 161, 'Steinway' = 162, 'Stuyvesant Heights' = 163, 'Stuyvesant Town-Cooper Village' = 164, 'Sunset Park East' = 165, 'Sunset Park West' = 166, 'Todt Hill-Emerson Hill-Heartland Village-Lighthouse Hill' = 167, 'Turtle Bay-East Midtown' = 168, 'University Heights-Morris Heights' = 169, 'Upper East Side-Carnegie Hill' = 170, 'Upper West Side' = 171, 'Van Cortlandt Village' = 172, 'Van Nest-Morris Park-Westchester Square' = 173, 'Washington Heights North' = 174, 'Washington Heights South' = 175, 'West Brighton' = 176, 'West Concourse' = 177, 'West Farms-Bronx River' = 178, 'West New Brighton-New Brighton-St. George' = 179, 'West Village' = 180, 'Westchester-Unionport' = 181, 'Westerleigh' = 182, 'Whitestone' = 183, 'Williamsbridge-Olinville' = 184, 'Williamsburg' = 185, 'Windsor Terrace' = 186, 'Woodhaven' = 187, 'Woodlawn-Wakefield' = 188, 'Woodside' = 189, 'Yorkville' = 190, 'park-cemetery-etc-Bronx' = 191, 'park-cemetery-etc-Brooklyn' = 192, 'park-cemetery-etc-Manhattan' = 193, 'park-cemetery-etc-Queens' = 194, 'park-cemetery-etc-Staten Island' = 195) AS pickup\_ntaname,

toUInt16(ifNull(pickup\_puma, '0')) AS pickup\_puma,

assumeNotNull(dropoff\_nyct2010\_gid) AS dropoff\_nyct2010\_gid,  
toFloat32(ifNull(dropoff\_ctlabel, '0')) AS dropoff\_ctlabel,  
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\_boroc2010, '0000000'), 7) AS dropoff\_boroc2010,  
CAST(assumeNotNull(ifNull(dropoff\_cdeligibil, ' ')) AS Enum8(' ' = 0, 'E' = 1, 'I' = 2)) AS dropoff\_cdeligibil,  
toFixedString(ifNull(dropoff\_ntacode, '0000'), 4) AS dropoff\_ntacode,

CAST(assumeNotNull(dropoff\_ntaname) AS Enum16(' ' = 0, 'Airport' = 1, 'Allerton-Pelham Gardens' = 2, 'Annadale-Huguenot-Prince's Bay-Eltingville' = 3, 'Arden Heights' = 4, 'Astoria' = 5, 'Auburndale' = 6, 'Baisley Park' = 7, 'Bath Beach' = 8, 'Battery Park City-Lower Manhattan' = 9, 'Bay Ridge' = 10, 'Bayside-Bayside Hills' = 11, 'Bedford' = 12, 'Bedford Park-Fordham North' = 13, 'Bellerose' = 14, 'Belmont' = 15, 'Bensonhurst East' = 16, 'Bensonhurst West' = 17, 'Borough Park' = 18, 'Breezy Point-Belle Harbor-Rockaway Park-Broad Channel' = 19, 'Briarwood-Jamaica Hills' = 20, 'Brighton Beach' = 21, 'Bronxdale' = 22, 'Brooklyn Heights-Cobble Hill' = 23, 'Brownsville' = 24, 'Bushwick North' = 25, 'Bushwick South' = 26, 'Cambria Heights' = 27, 'Canarsie' = 28, 'Carroll Gardens-Columbia Street-Red Hook' = 29, 'Central Harlem North-Polo Grounds' = 30, 'Central Harlem South' = 31, 'Charleston-Richmond Valley-Tottenville' = 32, 'Chinatown' = 33, 'Claremont-Bathgate' = 34, 'Clinton' = 35, 'Clinton Hill' = 36, 'Co-op City' = 37, 'College Point' = 38, 'Corona' = 39, 'Crotona Park East' = 40, 'Crown Heights North' = 41, 'Crown Heights South' = 42, 'Cypress Hills-City Line' = 43, 'DUMBO-Vinegar Hill-Downtown Brooklyn-Boerum Hill' = 44, 'Douglas Manor-Douglaslaston-Little Neck' = 45, 'Dyker Heights' = 46, 'East Concourse-Concourse Village' = 47, 'East Elmhurst' = 48, 'East Flatbush-Farragut' = 49, 'East Flushing' = 50, 'East Harlem North' = 51, 'East Harlem South' = 52, 'East New York' = 53, 'East New York (Pennsylvania Ave)' = 54, 'East Tremont' = 55, 'East Village' = 56, 'East Williamsburg' = 57, 'Eastchester-Edenwald-Baychester' = 58, 'Elmhurst' = 59, 'Elmhurst-Maspeth' = 60, 'Erasmus' = 61, 'Far Rockaway-Bayswater' = 62, 'Flatbush' = 63, 'Flatlands' = 64, 'Flushing' = 65, 'Fordham South' = 66, 'Forest Hills' = 67, 'Fort Greene' = 68, 'Fresh Meadows-Utopia' = 69, 'Ft. Totten-Bay Terrace-Clearview' = 70, 'Georgetown-Marine Park-Bergen Beach-Mill Basin' = 71, 'Glen Oaks-Floral Park-New Hyde Park' = 72, 'Glendale' = 73, 'Gramercy' = 74, 'Grasmere-Arrochar-Ft. Wadsworth' = 75, 'Gravesend' = 76, 'Great Kills' = 77, 'Greenpoint' = 78, 'Grymes Hill-Clifton-Fox Hills' = 79, 'Hamilton Heights' = 80, 'Hammels-Arverne-Edgemere' = 81, 'Highbridge' = 82, 'Hollis' = 83, 'Homecrest' = 84, 'Hudson Yards-Chelsea-Flatiron-Union Square' = 85, 'Hunters Point-Sunnyside-West Maspeth' = 86, 'Hunts Point' = 87, 'Jackson Heights' = 88, 'Jamaica' = 89, 'Jamaica Estates-Holliswood' = 90, 'Kensington-Ocean Parkway' = 91, 'Kew Gardens' = 92, 'Kew Gardens Hills' = 93, 'Kingsbridge Heights' = 94, 'Laurelton' = 95, 'Lenox Hill-Roosevelt Island' = 96, 'Lincoln Square' = 97, 'Lindenwood-Howard Beach' = 98, 'Longwood' = 99, 'Lower East Side' = 100, 'Madison' = 101, 'Manhattanville' = 102, 'Marble Hill-Inwood' = 103, 'Mariner's Harbor-Arlington-Port Ivory-Graniteville' = 104, 'Maspeth' = 105, 'Melrose South-Mott Haven North' = 106, 'Middle Village' = 107, 'Midtown-Midtown South' = 108, 'Midwood' = 109, 'Morningside Heights' = 110, 'Morrisania-Melrose' = 111, 'Mott Haven-Port Morris' = 112, 'Mount Hope' = 113, 'Murray Hill' = 114, 'Murray Hill-Kips Bay' = 115, 'New Brighton-Silver Lake' = 116, 'New Dorp-Midland Beach' = 117, 'New Springville-Bloomfield-Travis' = 118, 'North Corona' = 119, 'North Riverdale-

```

Fieldston-Riverdale' = 120, 'North Side-South Side' = 121, 'Norwood' = 122, 'Oakland Gardens' = 123, 'Oakwood-
Oakwood Beach' = 124, 'Ocean Hill' = 125, 'Ocean Parkway South' = 126, 'Old Astoria' = 127, 'Old Town-Dongan
Hills-South Beach' = 128, 'Ozone Park' = 129, 'Park Slope-Gowanus' = 130, 'Parkchester' = 131, 'Pelham Bay-Country
Club-City Island' = 132, 'Pelham Parkway' = 133, 'Pomonok-Flushing Heights-Hillcrest' = 134, 'Port Richmond' = 135,
'Prospect Heights' = 136, 'Prospect Lefferts Gardens-Wingate' = 137, 'Queens Village' = 138, 'Queensboro Hill' = 139,
'Queensbridge-Ravenswood-Long Island City' = 140, 'Rego Park' = 141, 'Richmond Hill' = 142, 'Ridgewood' = 143,
'Rikers Island' = 144, 'Rosedale' = 145, 'Rossville-Woodrow' = 146, 'Rugby-Remsen Village' = 147, 'Schuylerville-
Throgs Neck-Edgewater Park' = 148, 'Seagate-Coney Island' = 149, 'Sheepshead Bay-Gerritsen Beach-Manhattan
Beach' = 150, 'SoHo-TriBeCa-Civic Center-Little Italy' = 151, 'Soundview-Bruckner' = 152, 'Soundview-Castle Hill-
Clason Point-Harding Park' = 153, 'South Jamaica' = 154, 'South Ozone Park' = 155, 'Springfield Gardens North' = 156,
'Springfield Gardens South-Brooklyn' = 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

```

Это занимает 3030 секунд со скоростью около 428 тысяч строк в секунду.

Для более короткого времени загрузки, можно создать таблицу с движком Log вместо MergeTree. В этом случае загрузка отработает быстрее, чем за 200 секунд.

Таблица заняла 126 Гб дискового пространства.

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

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

Между прочим, на MergeTree можно запустить запрос OPTIMIZE. Но это не обязательно, всё будет в порядке и без этого.

## Скачивание готовых партиций

```
$ curl -O https://datasets.clickhouse.com/trips_mergetree/partitions/trips_mergetree.tar
$ tar xvf trips_mergetree.tar -C /var/lib/clickhouse # путь к папке с данными ClickHouse
$ # убедитесь, что установлены корректные права доступа на файлы
$ sudo service clickhouse-server restart
$ clickhouse-client --query "SELECT COUNT(*) FROM datasets.trips_mergetree"
```

### Info

Если вы собираетесь выполнять запросы, приведенные ниже, то к имени таблицы

нужно добавить имя базы, `datasets.trips_mergetree`.

## Результаты на одном сервере

Q1:

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

0.490 секунд.

Q2:

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

1.224 секунд.

Q3:

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

2.104 секунд.

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 секунд.

Использовался следующий сервер:

Два Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz, в сумме 16 физических ядер,  
128 GiB RAM,  
8x6 TB HD на программном RAID-5

Время выполнения — лучшее из трех запусков.

На самом деле начиная со второго запуска, запросы читают данные из кеша страниц файловой системы. Никакого дальнейшего кеширования не происходит: данные зачитываются и обрабатываются при каждом запуске.

Создание таблицы на 3 серверах:

На каждом сервере:

```
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" = 0, 'Manhattan' = 1, 'Bronx' = 2, 'Brooklyn' = 3, 'Queens' = 4, 'Staten Island' = 5), pickup_ct2010 fixedString(6), pickup_boroct2010 fixedString(7), pickup_cdeligibil Enum8('0' = 0, 'E' = 1, 'I' = 2), pickup_ntacode fixedString(4), pickup_ntaname Enum16("0" = 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-Douglaston-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\_borocat2010 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, 'Belleroose' = 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-Douglaston-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)
```

На исходном сервере:

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

Следующим запрос перераспределит данные:

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

Это занимает 2454 секунд.

На трёх серверах:

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

Никакого сюрприза, так как запросы масштабируются линейно.

Также у нас есть результаты с кластера из 140 серверов:

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

В этом случае, время выполнения запросов определяется в первую очередь сетевыми задержками. Мы выполняли запросы с помощью клиента, расположенного в data-центре Яндекса в Мянтсяля (Финляндия), на кластер в России, что добавляет порядка 20 мс задержки.

## Резюме

серверов	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

# Набор данных о воздушном движении OpenSky Network 2020

"Данные в этом наборе получены и отфильтрованы из полного набора данных OpenSky, чтобы проиллюстрировать развитие воздушного движения во время пандемии COVID-19. Набор включает в себя все рейсы, которые видели более 2500 участников сети с 1 января 2019 года. Дополнительные данные будут периодически включаться в набор данных до окончания пандемии COVID-19".

Источник: <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>

## Загрузите набор данных

Выполните команду:

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

Загрузка займет около 2 минут при хорошем подключении к интернету. Будет загружено 30 файлов общим размером 4,3 ГБ.

## Создайте таблицу

```
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);
```

## Импортируйте данные в ClickHouse

Загрузите данные в ClickHouse параллельными потоками:

```
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"'
```

- Список файлов передаётся (`ls -1 flightlist_*.csv.gz`) в `xargs` для параллельной обработки.

- `xargs -P100` указывает на возможность использования до 100 параллельных обработчиков, но поскольку у нас всего 30 файлов, то количество обработчиков будет всего 30.
- Для каждого файла `xargs` будет запускать скрипт с `bash -c`. Сценарий имеет подстановку в виде {}, а команда `xargs` заменяет имя файла на указанные в подстановке символы (мы указали это для `xargs` с помощью `-I{}`).
- Скрипт распакует файл (`gzip -c -d "{}"`) в стандартный вывод (параметр `-c`) и перенаправит его в `clickhouse-client`.
- Чтобы распознать формат ISO-8601 со смещениями часовых поясов в полях типа `DateTime`, указывается параметр парсера `--date_time_input_format best_effort`.

В итоге: клиент `clickhouse` добавит данные в таблицу `opensky`. Входные данные импортируются в формате `CSVWithNames`.

Загрузка параллельными потоками займёт около 24 секунд.

Также вы можете использовать вариант последовательной загрузки:

```
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
```

## Проверьте импортированные данные

Запрос:

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

Результат:

```
count()
66010819
```

Убедитесь, что размер набора данных в ClickHouse составляет всего 2,66 GiB.

Запрос:

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

Результат:

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

## Примеры

Общее пройденное расстояние составляет 68 миллиардов километров.

Запрос:

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

Результат:

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

Средняя дальность полета составляет около 1000 км.

Запрос:

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

Результат:

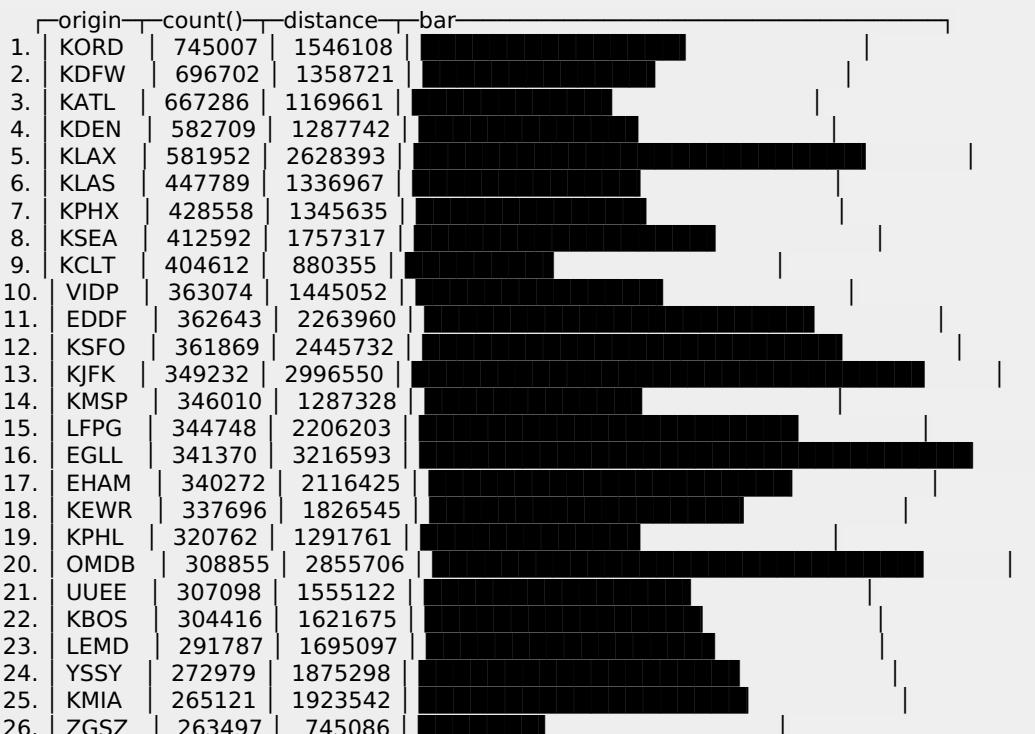
```
avg(geoDistance(longitude_1, latitude_1, longitude_2, latitude_2))—  
1041090.6465708319
```

## Наиболее загруженные аэропорты в указанных координатах и среднее пройденное расстояние

Запрос:

```
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;
```

Результат:



27.	EDDM	256691	1361453					
28.	WMKK	254264	1626688					
29.	CYYZ	251192	2175026					
30.	KLGA	248699	1106935					
31.	VHHH	248473	3457658					
32.	RJTT	243477	1272744					
33.	KBWI	241440	1187060					
34.	KIAD	239558	1683485					
35.	KIAH	234202	1538335					
36.	KFLL	223447	1464410					
37.	KDAL	212055	1082339					
38.	KDCA	207883	1013359					
39.	LIRF	207047	1427965					
40.	PANC	206007	2525359					
41.	LTFJ	205415	860470					
42.	KDTW	204020	1106716					
43.	VABB	201679	1300865					
44.	OTHH	200797	3759544					
45.	KMDW	200796	1232551					
46.	KSAN	198003	1495195					
47.	KPDX	197760	1269230					
48.	SBGR	197624	2041697					
49.	VOBL	189011	1040180					
50.	LEBL	188956	1283190					
51.	YBBN	188011	1253405					
52.	LSZH	187934	1572029					
53.	YMML	187643	1870076					
54.	RCTP	184466	2773976					
55.	KSNA	180045	778484					
56.	EGKK	176420	1694770					
57.	LOWW	176191	1274833					
58.	UUDD	176099	1368226					
59.	RKSI	173466	3079026					
60.	EKCH	172128	1229895					
61.	KOAK	171119	1114447					
62.	RPLL	170122	1440735					
63.	KRDU	167001	830521					
64.	KAUS	164524	1256198					
65.	KBNA	163242	1022726					
66.	KSDF	162655	1380867					
67.	ENGM	160732	910108					
68.	LIMC	160696	1564620					
69.	KSJC	159278	1081125					
70.	KSTL	157984	1026699					
71.	UUWW	156811	1261155					
72.	KIND	153929	987944					
73.	ESSA	153390	1203439					
74.	KMCO	153351	1508657					
75.	KDVT	152895	74048					
76.	VTBS	152645	2255591					
77.	CYVR	149574	2027413					
78.	EIDW	148723	1503985					
79.	LFPO	143277	1152964					
80.	EGSS	140830	1348183					
81.	KAPA	140776	420441					
82.	KHOU	138985	1068806					
83.	KTPA	138033	1338223					
84.	KFFZ	137333	55397					
85.	NZAA	136092	1581264					
86.	YPPH	133916	1271550					
87.	RJBB	133522	1805623					
88.	EDDL	133018	1265919					
89.	ULLI	130501	1197108					
90.	KIWA	127195	250876					
91.	KTEB	126969	1189414					
92.	VOMM	125616	1127757					
93.	LSGG	123998	1049101					
94.	LPPT	122733	1779187					
95.	WSSS	120493	3264122					
96.	EBBR	118539	1579939					
97.	VTBD	118107	661627					
98.	KVNY	116326	692960					
99.	EDDT	115122	941740					
100.	EFHK	114860	1629143					

# Номера рейсов из трех крупных аэропортов Москвы, еженедельно

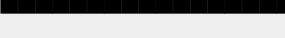
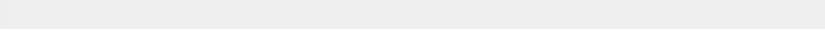
Запрос:

```
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;
```

Результат:



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	
40.	2019-09-30	6880	
41.	2019-10-07	6518	
42.	2019-10-14	6688	
43.	2019-10-21	6667	
44.	2019-10-28	6303	
45.	2019-11-04	6298	
46.	2019-11-11	6137	
47.	2019-11-18	6051	
48.	2019-11-25	5820	
49.	2019-12-02	5942	
50.	2019-12-09	4891	
51.	2019-12-16	5682	
52.	2019-12-23	6111	
53.	2019-12-30	5870	
54.	2020-01-06	5953	
55.	2020-01-13	5698	
56.	2020-01-20	5339	
57.	2020-01-27	5566	
58.	2020-02-03	5801	
59.	2020-02-10	5692	
60.	2020-02-17	5912	
61.	2020-02-24	6031	
62.	2020-03-02	6105	
63.	2020-03-09	5823	
64.	2020-03-16	4659	
65.	2020-03-23	3720	

66.	2020-03-30	1720	
67.	2020-04-06	849	
68.	2020-04-13	710	
69.	2020-04-20	725	
70.	2020-04-27	920	
71.	2020-05-04	859	
72.	2020-05-11	1047	
73.	2020-05-18	1135	
74.	2020-05-25	1266	
75.	2020-06-01	1793	
76.	2020-06-08	1979	
77.	2020-06-15	2297	
78.	2020-06-22	2788	
79.	2020-06-29	3389	
80.	2020-07-06	3545	
81.	2020-07-13	3569	
82.	2020-07-20	3784	
83.	2020-07-27	3960	
84.	2020-08-03	4323	
85.	2020-08-10	4581	
86.	2020-08-17	4791	
87.	2020-08-24	4928	
88.	2020-08-31	4687	
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	

## Online Playground

Вы можете протестировать другие запросы к этому набору данным с помощью интерактивного ресурса [Online Playground](#). Например, [вот так](#). Однако обратите внимание, что здесь нельзя создавать временные таблицы.

# Набор данных о стоимости недвижимости в Великобритании

Набор содержит данные о стоимости недвижимости в Англии и Уэльсе. Данные доступны с 1995 года.

Размер набора данных в несжатом виде составляет около 4 GiB, а в ClickHouse он займет около 278 MiB.

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

Описание полей таблицы: <https://www.gov.uk/guidance/about-the-price-paid-data>

Набор содержит данные HM Land Registry data © Crown copyright and database right 2021. Эти данные лицензированы в соответствии с Open Government Licence v3.0.

## Загрузите набор данных

Выполните команду:

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

Загрузка займет около 2 минут при хорошем подключении к интернету.

## Создайте таблицу

```
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);
```

## Обработайте и импортируйте данные

В этом примере используется `clickhouse-local` для предварительной обработки данных и `clickhouse-client` для импорта данных.

Указывается структура исходных данных CSV-файла и запрос для предварительной обработки данных с помощью `clickhouse-local`.

Предварительная обработка включает:

- разделение почтового индекса на два разных столбца `postcode1` и `postcode2`, что лучше подходит для хранения данных и выполнения запросов к ним;
- преобразование поля `time` в дату, поскольку оно содержит только время 00:00;
- поле `UUID` игнорируется, потому что оно не будет использовано для анализа;
- преобразование полей `type` и `duration` в более читаемые поля типа `Enum` с помощью функции `transform`;
- преобразование полей `is_new` и `category` из односимвольной строки (Y/N и A/B) в поле `UInt8` со значениями 0 и 1 соответственно.

Обработанные данные передаются в `clickhouse-client` и импортируются в таблицу ClickHouse потоковым способом.

```
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"
```

Выполнение запроса займет около 40 секунд.

## Проверьте импортированные данные

Запрос:

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

Результат:

```
count()
26321785
```

Размер набора данных в ClickHouse составляет всего 278 MiB, проверьте это.

Запрос:

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

Результат:

```
formatReadableSize(total_bytes)
278.80 MiB
```

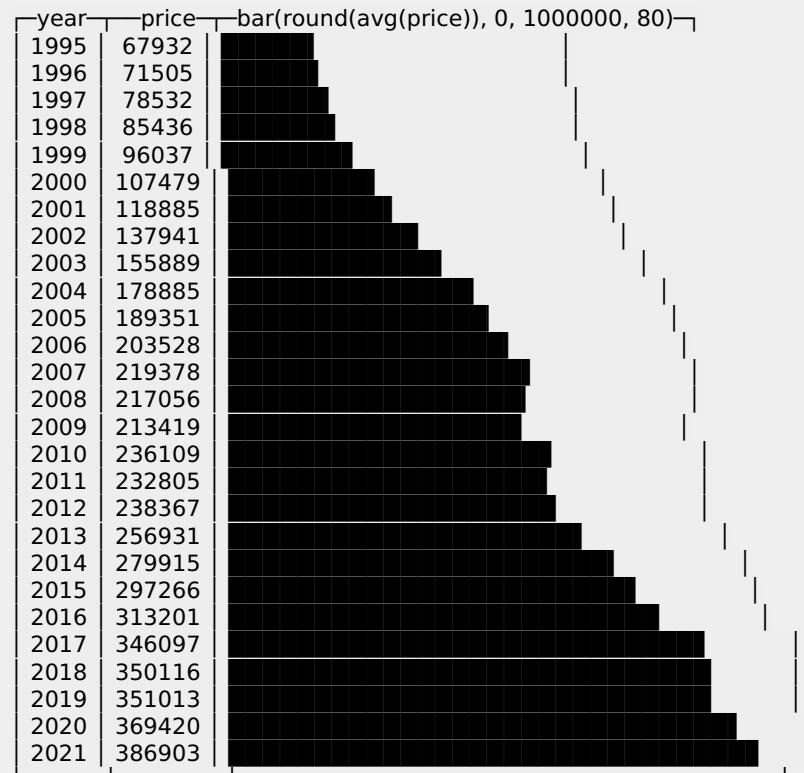
## Примеры запросов

### Запрос 1. Средняя цена за год

Запрос:

```
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;
```

Результат:

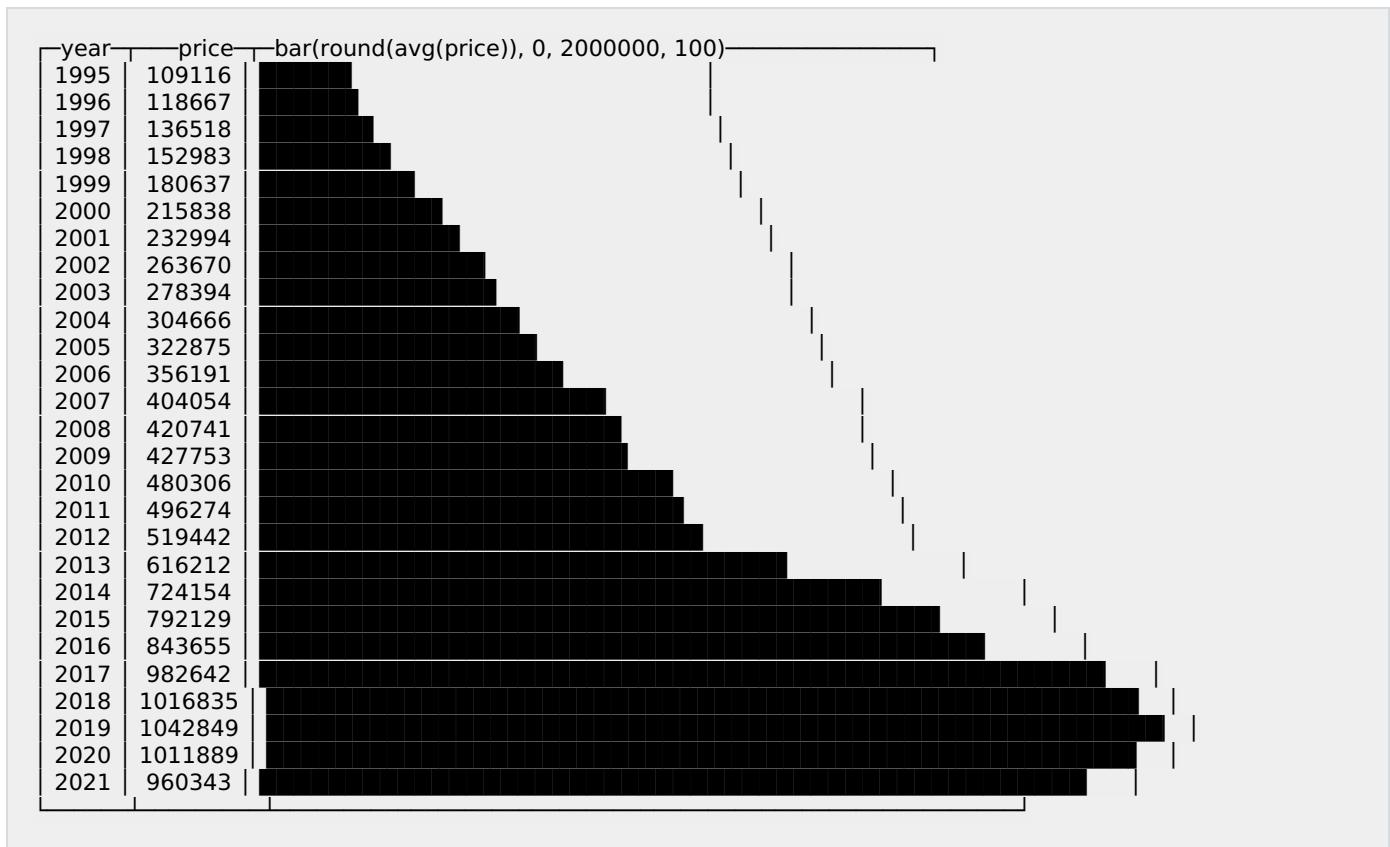


### Запрос 2. Средняя цена за год в Лондоне

Запрос:

```
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;
```

Результат:



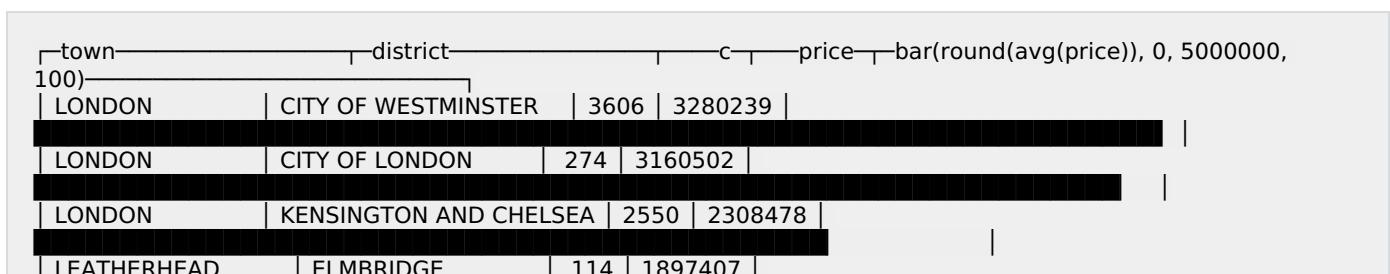
Что-то случилось в 2013 году. Я понятия не имею. Может быть, вы имеете представление о том, что произошло в 2020 году?

### Запрос 3. Самые дорогие районы

Запрос:

```
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;
```

Результат:



LONDON	CAMDEN	3033	1805404	
VIRGINIA WATER	RUNNYMEDE	156	1753247	
WINDLESHAM	SURREY HEATH	108	1677613	
THORNTON HEATH	CROYDON	546	1671721	
BARNET	ENFIELD	124	1505840	
COBHAM	ELMBRIDGE	387	1237250	
LONDON	ISLINGTON	2668	1236980	
OXFORD	SOUTH OXFORDSHIRE	321	1220907	
LONDON	RICHMOND UPON THAMES	704	1215551	
LONDON	HOUNSLOW	671	1207493	
ASCOT	WINDSOR AND MAIDENHEAD	407	1183299	
BEACONSFIELD	BUCKINGHAMSHIRE	330	1175615	
RICHMOND	RICHMOND UPON THAMES	874	1110444	
LONDON	HAMMERSMITH AND FULHAM	3086	1053983	
SURBITON	ELMBRIDGE	100	1011800	
RADLETT	HERTSMERE	283	1011712	
SALCOMBE	SOUTH HAMS	127	1011624	
WEYBRIDGE	ELMBRIDGE	655	1007265	
ESHER	ELMBRIDGE	485	986581	
LEATHERHEAD	GUILDFORD	202	977320	
BURFORD	WEST OXFORDSHIRE	111	966893	
BROCKENHURST	NEW FOREST	129	956675	
HINDHEAD	WAVERLEY	137	953753	
GERRARDS CROSS	BUCKINGHAMSHIRE	419	951121	
EAST MOLESEY	ELMBRIDGE	192	936769	
CHALFONT ST GILES	BUCKINGHAMSHIRE	146	925515	
LONDON	TOWER HAMLETS	4388	918304	
OLNEY	MILTON KEYNES	235	910646	
HENLEY-ON-THAMES	SOUTH OXFORDSHIRE	540	902418	
LONDON	SOUTHWARK	3885	892997	
KINGSTON UPON THAMES	KINGSTON UPON THAMES	960	885969	
CRANBROOK	EALING	2658	871755	
	TUNBRIDGE WELLS	431	862348	
LONDON	MERTON	2099	859118	
BELVEDERE	BEXLEY	346	842423	
GUILDFORD	WAVERLEY	143	841277	
HARPENDEN	ST ALBANS	657	841216	
LONDON	HACKNEY	3307	837090	
LONDON	WANDSWORTH	6566	832663	
MAIDENHEAD	BUCKINGHAMSHIRE	123	824299	
KINGS LANGLEY	DACORUM	145	821331	
BERKHAMSTED	DACORUM	543	818415	

GREAT MISSENDEN	BUCKINGHAMSHIRE	226	802807	
BILLINGSHURST	CHICHESTER	144	797829	
WOKING	GUILDFORD	176	793494	
STOCKBRIDGE	TEST VALLEY	178	793269	
EPSOM	REIGATE AND BANSTEAD	172	791862	
TONBRIDGE	TUNBRIDGE WELLS	360	787876	
TEDDINGTON	RICHMOND UPON THAMES	595	786492	
TWICKENHAM	RICHMOND UPON THAMES	1155	786193	
LYNDHURST	NEW FOREST	102	785593	
LONDON	LAMBETH	5228	774574	
LONDON	BARNET	3955	773259	
OXFORD	VALE OF WHITE HORSE	353	772088	
TONBRIDGE	MAIDSTONE	305	770740	
LUTTERWORTH	HARBOROUGH	538	768634	
WOODSTOCK	WEST OXFORDSHIRE	140	766037	
MIDHURST	CHICHESTER	257	764815	
MARLOW	BUCKINGHAMSHIRE	327	761876	
LONDON	NEWHAM	3237	761784	
ALDERLEY EDGE	CHESTER EAST	178	757318	
LUTON	CENTRAL BEDFORDSHIRE	212	754283	
PETWORTH	CHICHESTER	154	754220	
ALRESFORD	WINCHESTER	219	752718	
POTTERS BAR	WELWYN HATFIELD	174	748465	
HASLEMERE	CHICHESTER	128	746907	
TADWORTH	REIGATE AND BANSTEAD	502	743252	
THAMES DITTON	ELMBRIDGE	244	741913	
REIGATE	REIGATE AND BANSTEAD	581	738198	
BOURNE END	BUCKINGHAMSHIRE	138	735190	
SEVENOAKS	SEVENOAKS	1156	730018	
OXTED	TANDRIDGE	336	729123	
INGATESTONE	BRENTWOOD	166	728103	
LONDON	BRENT	2079	720605	
LONDON	HARINGEY	3216	717780	
PURLEY	CROYDON	575	716108	
WELWYN	WELWYN HATFIELD	222	710603	
RICKMANSWORTH	THREE RIVERS	798	704571	
BANSTEAD	REIGATE AND BANSTEAD	401	701293	
CHIGWELL	EPPING FOREST	261	701203	
PINNER	HARROW	528	698885	
HASLEMERE	WAVERLEY	280	696659	
SLOUGH	BUCKINGHAMSHIRE	396	694917	
WALTON-ON-THAMES	ELMBRIDGE	946	692395	
READING	SOUTH OXFORDSHIRE	318	691988	
NORTHWOOD	HILLINGDON	271	690643	
FELTHAM	HOUNSLAW	763	688595	
ASHTead	MOLE VALLEY	303	687923	
BARNET	BARNET	975	686980	
WOKING	SURREY HEATH	283	686669	
MALMESBURY	WILTSHIRE	323	683324	
AMERSHAM	BUCKINGHAMSHIRE	496	680962	
CHISLEHURST	BROMLEY	430	680209	
HYTHE	FOLKESTONE AND HYTHE	490	676908	

MAYFIELD ASCOT	WEALDEN   BRACKNELL FOREST	101   676210   [REDACTED]	168   676004   [REDACTED]	[REDACTED]
-------------------	-------------------------------	---------------------------	---------------------------	------------

Ускорьте запросы с помощью проекций

**Проекции** позволяют повысить скорость запросов за счет хранения предварительно агрегированных данных.

## Создайте проекцию

Создайте агрегирующую проекцию по параметрам `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
);
```

Заполните проекцию для текущих данных (иначе проекция будет создана только для добавляемых данных):

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

## Проверьте производительность

Давайте выполним те же 3 запроса.

**Включите поддержку проекций!**

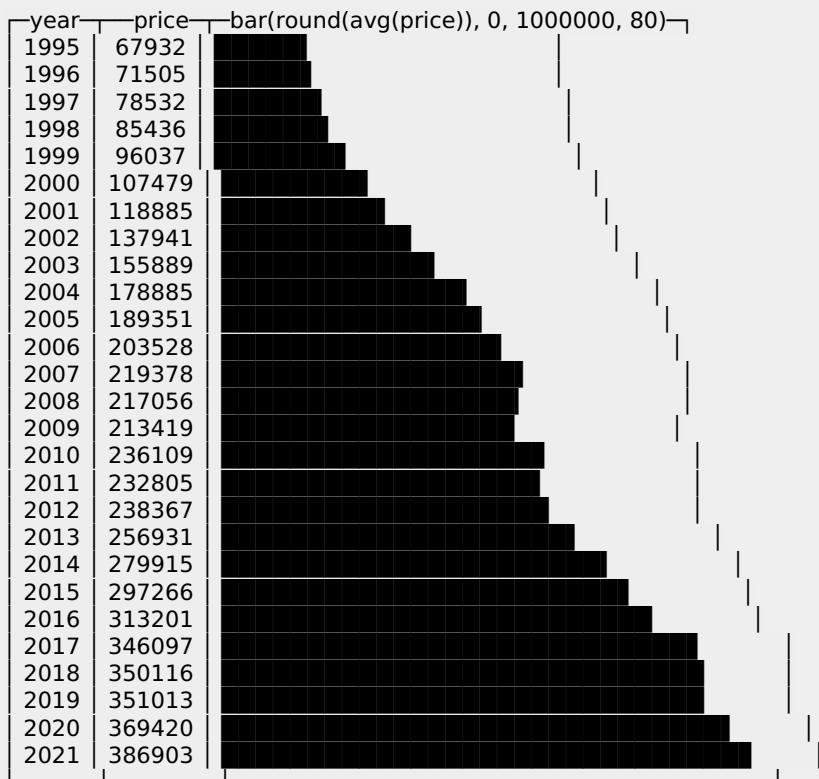
SET allow experimental projection optimization = 1;

## Запрос 1. Средняя цена за год

## Запрос:

```
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;
```

## Результат:

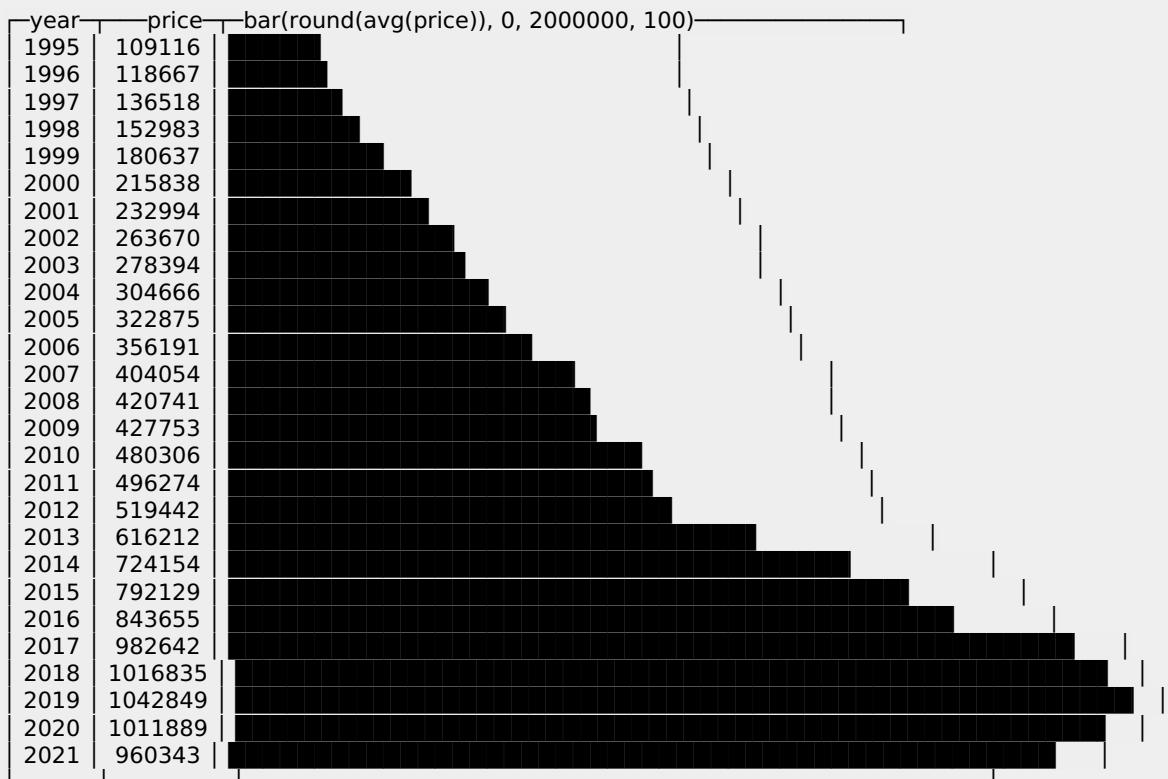


## Запрос 2. Средняя цена за год в Лондоне

Запрос:

```
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;
```

Результат:



### Запрос 3. Самые дорогие районы

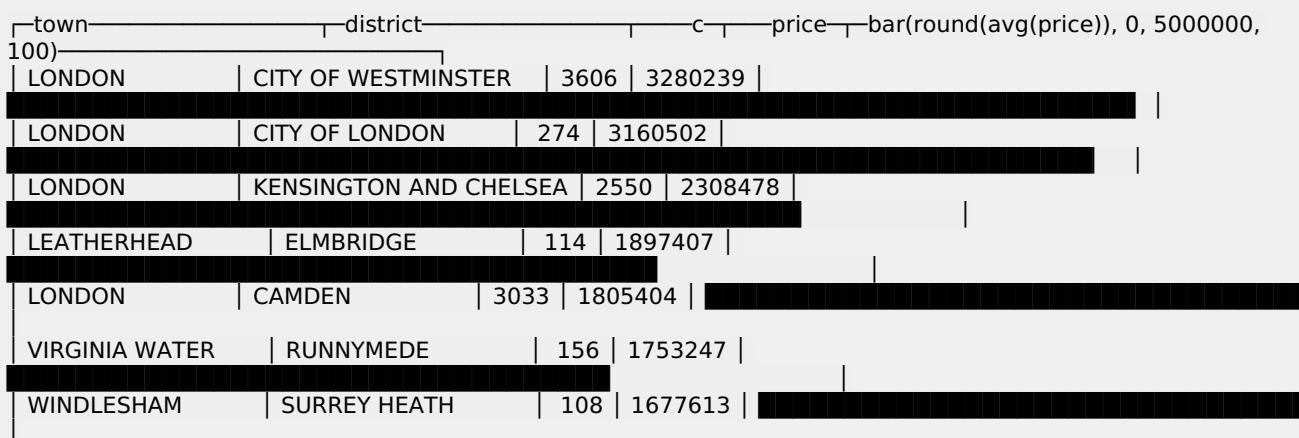
Условие (date >= '2020-01-01') необходимо изменить, чтобы оно соответствовало проекции (toYear(date) >= 2020).

Запрос:

```

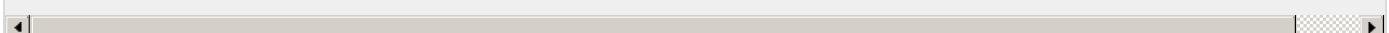
SELECT
  town,
  district,
  count() AS c,
  round(avg(price)) AS price,
  bar(price, 0, 5000000, 100)
FROM uk_price_paid
WHERE toYear(date) >= 2020
GROUP BY
  town,
  district
HAVING c >= 100
ORDER BY price DESC
LIMIT 100;
    
```

Результат:



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]



## Резюме

Все три запроса работают намного быстрее и читают меньшее количество строк.

### 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.)
```

## Online Playground

Этот набор данных доступен в [Online Playground](#).

## OnTime

Этот датасет может быть получен двумя способами:

- импорт из сырых данных;
- скачивание готовых партиций.

## Импорт из сырых данных

Скачивание данных (из <https://github.com/Percona-Lab/ontime-airline-performance/blob/master/download.sh>):

```
for s in `seq 1987 2018`  
do  
for m in `seq 1 12`  
do  
wget  
https://transtats.bts.gov/PREZIP/On_Time_Reportng_Carrier_On_Time_Performance_1987_present_${s}_${m}.zip  
done  
done
```

Создание таблицы:

```
CREATE TABLE `ontime`  
(  
`Year`          UInt16,  
`Quarter`       UInt8,  
`Month`         UInt8,  
`DayofMonth`    UInt8,  
`DayOfWeek`     UInt8,  
`FlightDate`    Date,  
`Reporting_Airline` String,  
`DOT_ID_Reportng_Airline` Int32,  
`IATA_CODE_Reportng_Airline` String,  
`Tail_Number`   Int32
```

```

`FlightNumber`           Int32,
`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`     String,
`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

```

```

`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_Reporting_Airline, FlightDate)
SETTINGS index_granularity = 8192;

```

Загрузка данных:

```
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 ontme FORMAT CSVWithNames'"
```

## Скачивание готовых партиций

```
$ curl -O https://datasets.clickhouse.com/ontime/partitions/ontime.tar
$ tar xvf ontme.tar -C /var/lib/clickhouse # путь к папке с данными ClickHouse
$ # убедитесь, что установлены корректные права доступа на файлы
$ sudo service clickhouse-server restart
$ clickhouse-client --query "SELECT COUNT(*) FROM datasets.ontime"
```

### Info

Если вы собираетесь выполнять запросы, приведенные ниже, то к имени таблицы

нужно добавить имя базы, datasets.ontime.

## Запросы:

Q0.

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

Q1. Количество полетов в день с 2000 по 2008 года

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

Q2. Количество полетов, задержанных более чем на 10 минут, с группировкой по дням неделе, за 2000-2008 года

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

Q3. Количество задержек по аэропортам за 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. Количество задержек по перевозчикам за 2007 год

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

Q5. Процент задержек по перевозчикам за 2007 год

```
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;
```

Более оптимальная версия того же запроса:

```
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. Предыдущий запрос за более широкий диапазон лет, 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;
```

Более оптимальная версия того же запроса:

```
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. Процент полетов, задержанных на более 10 минут, в разбивке по годам

```
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;
```

Более оптимальная версия того же запроса:

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

Q8. Самые популярные направления по количеству напрямую соединенных городов для различных диапазонов лет

```
SELECT DestCityName, uniqExact(OriginCityName) AS u F
ROM 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;
```

Бонус:

```
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;
```

Данный тест производительности был создан Вадимом Ткаченко, статьи по теме:

- <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>
- 

## Вышки сотовой связи

Источник этого набора данных (dataset) - самая большая в мире открытая база данных о сотовых вышках - [OpenCellid](#). К 2021-му году здесь накопилось более, чем 40 миллионов записей о сотовых вышках (GSM, LTE, UMTS, и т.д.) по всему миру с их географическими координатами и метаданными (код страны, сети, и т.д.).

OpenCellID Project имеет лицензию Creative Commons Attribution-ShareAlike 4.0 International License, и мы распространяем снэпшот набора данных по условиям этой же лицензии. После авторизации можно загрузить последнюю версию набора данных.

## Как получить набор данных

1. Загрузите снэпшот набора данных за февраль 2021 [отсюда](#) (729 MB).
2. Если нужно, проверьте полноту и целостность при помощи команды:

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

3. Распакуйте набор данных при помощи команды:

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

4. Создайте таблицу:

```
CREATE TABLE cell_towers  
(  
    radio Enum8(" = 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. Вставьте данные:

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

## Примеры

- Количество вышек по типам:

```
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.)

- Количество вышек по **мобильному коду страны (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.)

Можно увидеть, что по количеству вышек лидируют следующие страны: США, Германия, Россия.

Вы также можете создать **внешний словарь** в ClickHouse для того, чтобы расшифровать эти значения.

## Пример использования

Рассмотрим применение функции pointInPolygon.

- Создаем таблицу, в которой будем хранить многоугольники:

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

- Очертания Москвы выглядят приблизительно так ("Новая Москва" в них не включена):

```
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), (37.83846476321406, 55.729799917464675), (37.83835745269769, 55.72919751082619), (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), (37.82081726983639, 55.64640836412121), (37.820614174591, 55.64164525405531), (37.818908190475426, 55.6421883258084), (37.81717543386075, 55.6411249038471), (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, 55.58775595845419), (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), (37.4160720370478, 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), (37.369922155757884, 55.76735417953104), (37.369892695770275, 55.76823419316575), (37.370214730163575, 55.782312184391266), (37.370493611114505, 55.78436801120489), (37.37120164550783, 55.78596427165359), (37.37284851456452, 55.7874378183096), (37.37608325135799, 55.7886695054807), (37.3764587460632, 55.78947647305964), (37.37530000265506, 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), (37.46338399999984, 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), (37.57942799999998, 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), (37.83341438888553, 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. Проверяем, сколько сотовых вышек находится в Москве:

```
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.)

Вы можете протестировать другие запросы с помощью интерактивного ресурса [Playground](#).

Например, [вот так](#). Однако, обратите внимание, что здесь нельзя создавать временные таблицы.

## Набор данных публичной библиотеки Нью-Йорка "Что в меню?"

Набор данных создан Нью-Йоркской публичной библиотекой. Он содержит исторические данные о меню отелей, ресторанов и кафе с блюдами, а также их ценами.

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

Эти данные находятся в открытом доступе.

Данные взяты из архива библиотеки, и они могут быть неполными и сложными для статистического анализа. Тем не менее, это тоже очень интересно.

В наборе всего 1,3 миллиона записей о блюдах в меню — очень небольшой объем данных для ClickHouse, но это все равно хороший пример.

## Загрузите набор данных

Выполните команду:

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

При необходимости замените ссылку на актуальную ссылку с <http://menus.nypl.org/data>.

Размер архива составляет около 35 МБ.

# Распакуйте набор данных

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

Размер распакованных данных составляет около 150 МБ.

Данные нормализованы и состоят из четырех таблиц:

- `Menu` — информация о меню: название ресторана, дата, когда было просмотрено меню, и т.д.
- `Dish` — информация о блюдах: название блюда вместе с некоторыми характеристиками.
- `MenuPage` — информация о страницах в меню, потому что каждая страница принадлежит какому-либо меню.
- `MenuItem` — один из пунктов меню. Блюдо вместе с его ценой на какой-либо странице меню: ссылки на блюдо и страницу меню.

## Создайте таблицы

Для хранения цен используется тип данных `Decimal`.

```

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;

```

## Импортируйте данные

Импортируйте данные в ClickHouse, выполните команды:

```
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
```

Поскольку данные представлены в формате CSV с заголовком, используется формат [CSVWithNames](#).

Отключите `format_csv_allow_single_quotes`, так как для данных используются только двойные кавычки, а одинарные кавычки могут находиться внутри значений и не должны сбивать с толку CSV-парсер.

Отключите `input_format_null_as_default`, поскольку в данных нет значений `NULL`.

В противном случае ClickHouse попытается проанализировать последовательности `\N` и может перепутать с `\` в данных.

Настройка `date_time_input_format best_effort` позволяет анализировать поля [DateTime](#) в самых разных форматах. К примеру, будет распознан ISO-8601 без секунд: '2000-01-01 01:02'. Без этой настройки допускается только фиксированный формат даты и времени.

## Денормализуйте данные

Данные представлены в нескольких таблицах в [нормализованном виде](#).

Это означает, что вам нужно использовать условие объединения [JOIN](#), если вы хотите получить, например, названия блюд из пунктов меню.

Для типовых аналитических задач гораздо эффективнее работать с предварительно объединенными данными, чтобы не использовать [JOIN](#) каждый раз. Такие данные называются денормализованными.

Создайте таблицу `menu_item_denorm`, которая будет содержать все данные, объединенные вместе:

```
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;
```

## Проверьте загруженные данные

Запрос:

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

Результат:

```
count()
1329175 |
```

## Примеры запросов

### Усредненные исторические цены на блюда

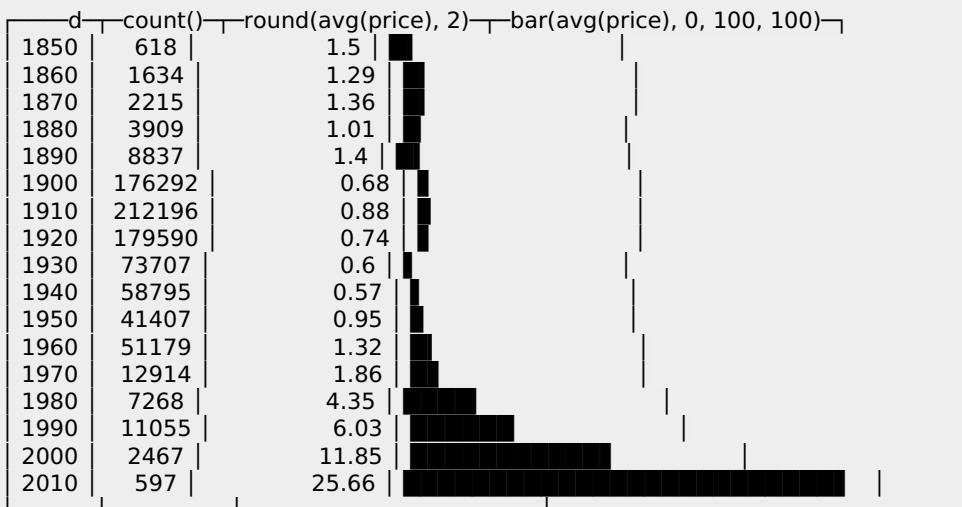
Запрос:

```

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;

```

Результат:



Просто не принимайте это всерьез.

## Цены на бургеры

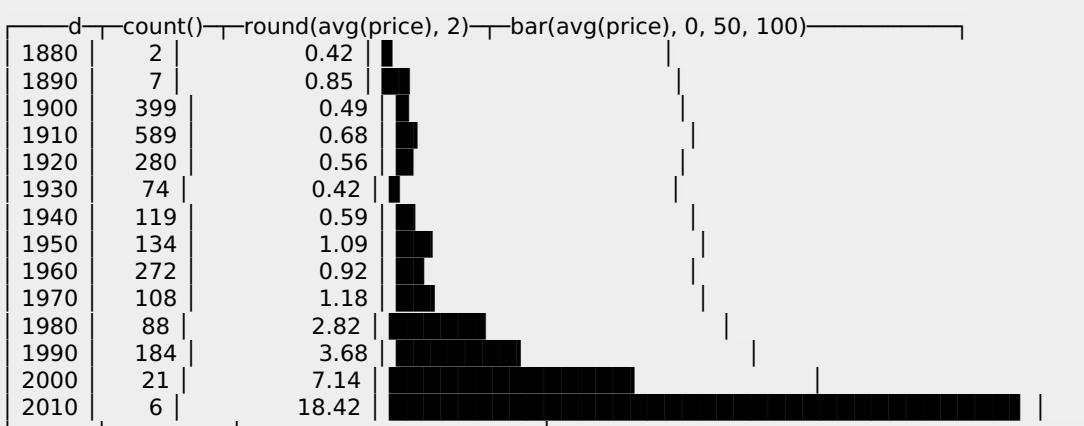
Запрос:

```

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;

```

Результат:

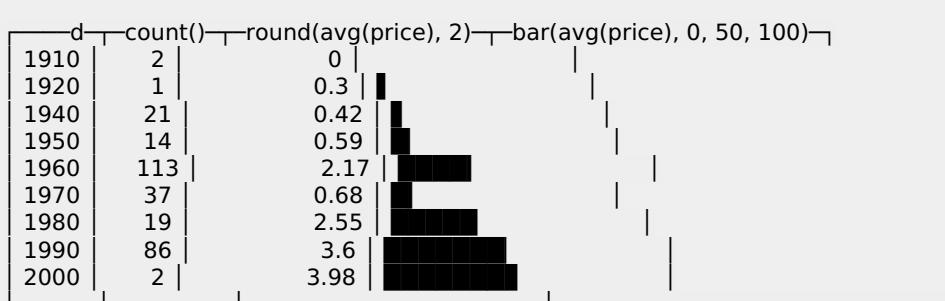


## Водка

Запрос:

```
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;
```

Результат:



Чтобы получить водку, мы должны написать ILIKE '%vodka%', и это хорошая идея.

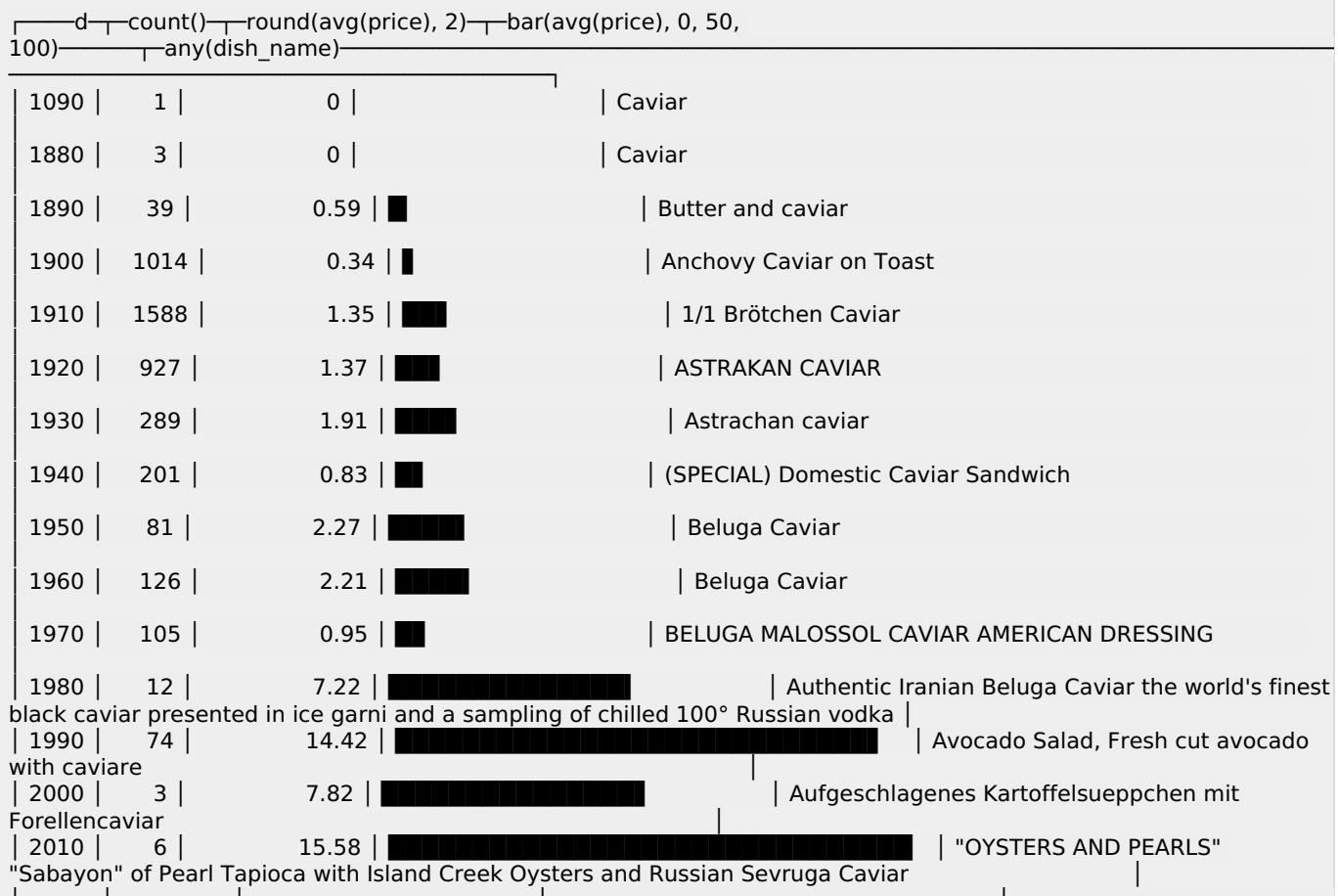
## Икра

Посмотрите цены на икру. Получите название любого блюда с икрой.

Запрос:

```
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;
```

Результат:



По крайней мере, есть икра с водкой. Очень мило.

## Online Playground

Этот набор данных доступен в интерактивном ресурсе [Online Playground](#).

## Интерфейсы

ClickHouse предоставляет два сетевых интерфейса (оба могут быть дополнительно обернуты в TLS для дополнительной безопасности):

- [HTTP](#), который задокументирован и прост для использования напрямую;
- [Native TCP](#), который имеет меньше накладных расходов.

В большинстве случаев рекомендуется использовать подходящий инструмент или библиотеку, а не напрямую взаимодействовать с ClickHouse по сути. Официально поддерживаемые Яндексом:

- [Консольный клиент](#);
- [JDBC-драйвер](#);
- [ODBC-драйвер](#);
- [C++ клиентская библиотека](#).

Существует также широкий спектр сторонних библиотек для работы с ClickHouse:

- Клиентские библиотеки;
- Библиотеки для интеграции;
- Визуальные интерфейсы.

## Клиент командной строки

ClickHouse предоставляет собственный клиент командной строки: `clickhouse-client`. Клиент поддерживает запуск с аргументами командной строки и с конфигурационными файлами. Подробнее читайте в разделе [Конфигурирование](#).

Клиент устанавливается пакетом `clickhouse-client` и запускается командой `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.

:)
```

Клиенты и серверы различных версий совместимы, однако если клиент старее сервера, то некоторые новые функции могут быть недоступны. Мы рекомендуем использовать одинаковые версии клиента и сервера. При подключении клиента к более новому серверу `clickhouse-client` выводит сообщение:

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

## Использование

Клиент может быть использован в интерактивном и не интерактивном (`batch`) режиме. Чтобы использовать `batch` режим, укажите параметр `query`, или отправьте данные в `stdin` (проверяется, что `stdin` - не терминал), или и то, и другое.

Аналогично HTTP интерфейсу, при использовании одновременно параметра `query` и отправке данных в `stdin`, запрос составляется из конкатенации параметра `query`, перевода строки и данных в `stdin`. Это удобно для больших `INSERT` запросов.

Примеры использования клиента для вставки данных:

```
$ 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";
```

В `batch` режиме в качестве формата данных по умолчанию используется формат `TabSeparated`. Формат может быть указан в запросе в секции `FORMAT`.

По умолчанию в `batch` режиме вы можете выполнить только один запрос. Чтобы выполнить несколько запросов из «скрипта», используйте параметр `--multิquery`. Это работает для всех запросов кроме `INSERT`. Результаты запросов выводятся подряд без дополнительных разделителей. Если нужно выполнить много запросов, вы можете запускать `clickhouse-client` отдельно на каждый запрос. Заметим, что запуск программы `clickhouse-client` может занимать десятки миллисекунд.

В интерактивном режиме вы получаете командную строку, в которую можно вводить запросы.

Если не указано `multiline` (по умолчанию):

Чтобы выполнить запрос, нажмите Enter. Точка с запятой на конце запроса необязательна. Чтобы ввести запрос, состоящий из нескольких строк, в конце строки поставьте символ обратного слеша \, тогда после нажатия Enter вы сможете ввести следующую строку запроса.

Если указан параметр `--multiline` (многострочный режим):

Чтобы выполнить запрос, завершите его точкой с запятой и нажмите Enter. Если в конце введённой строки не было точки с запятой, то вам предложат ввести следующую строчку запроса.

Исполняется только один запрос, поэтому всё, что введено после точки с запятой, игнорируется.

Вместо или после точки с запятой может быть указано \G. Это обозначает использование формата Vertical. В этом формате каждое значение выводится на отдельной строке, что удобно для широких таблиц. Столь необычная функциональность добавлена для совместимости с MySQL CLI.

Командная строка сделана на основе readline (и history) (или libedit, или без какой-либо библиотеки, в зависимости от сборки) - то есть, в ней работают привычные сочетания клавиш, а также присутствует история.

История пишется в `~/.clickhouse-client-history`.

По умолчанию используется формат вывода PrettyCompact (он поддерживает красивый вывод таблиц). Вы можете изменить формат вывода результатов запроса следующими способами: с помощью секции `FORMAT` в запросе, указав символ \G в конце запроса, используя аргументы командной строки `--format` или `--vertical` или с помощью конфигурационного файла клиента.

Чтобы выйти из клиента, нажмите Ctrl+D или наберите вместо запроса одно из: «exit», «quit», «logout», «учше», «йгше», «дщпщге», «exit;», «quit;», «logout;», «учшеж», «йгшеж», «дщпщгеж», «q», «Й», «q», «Q», «:q», «Й», «Й», «ЖЙ».

При выполнении запроса клиент показывает:

1. Прогресс выполнение запроса, который обновляется не чаще, чем 10 раз в секунду (по умолчанию). При быстрых запросах прогресс может не успеть отобразиться.
2. Отформатированный запрос после его парсинга - для отладки.
3. Результат в заданном формате.
4. Количество строк результата, прошедшее время, а также среднюю скорость выполнения запроса.

Вы можете прервать длинный запрос, нажав Ctrl+C. При этом вам всё равно придётся чуть-чуть подождать, пока сервер остановит запрос. На некоторых стадиях выполнения запрос невозможно прервать. Если вы не дождётесь и нажмёте Ctrl+C второй раз, то клиент будет завершён.

Клиент командной строки позволяет передать внешние данные (внешние временные таблицы) для выполнения запроса. Подробнее смотрите раздел «Внешние данные для обработки запроса».

## Запросы с параметрами

Вы можете создать запрос с параметрами и передавать в них значения из приложения. Это позволяет избежать форматирования запросов на стороне клиента, если известно, какие из параметров запроса динамически меняются. Например:

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

## Синтаксис запроса

Отформатируйте запрос обычным способом. Представьте значения, которые вы хотите передать из параметров приложения в запрос в следующем формате:

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

- `name` — идентификатор подстановки. В консольном клиенте его следует использовать как часть имени параметра `--param_<name> = value`.
- `data type` — **тип данных** значения. Например, структура данных (`integer`, (`'string'`, `integer`)) может иметь тип данных `Tuple(UInt8, Tuple(String, UInt8))` (**целочисленный** тип может быть и другим). В качестве параметра можно передать название столбца, таблицы и базы данных, в этом случае используется тип данных `Identifier`.

## Пример

```
$ 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"
```

# Конфигурирование

В `clickhouse-client` можно передавать различные параметры (все параметры имеют значения по умолчанию) с помощью:

- Командной строки.

Параметры командной строки переопределяют значения по умолчанию и параметры конфигурационных файлов.

- Конфигурационных файлов.

Параметры в конфигурационных файлах переопределяют значения по умолчанию.

## Параметры командной строки

- `--host`, `-h` — имя сервера, по умолчанию — `'localhost'`. Вы можете использовать как имя, так и IPv4 или IPv6 адрес.
- `--port` — порт для подключения, по умолчанию — `9000`. Обратите внимание: для HTTP-интерфейса и нативного интерфейса используются разные порты.
- `--user`, `-u` — имя пользователя, по умолчанию — `'default'`.
- `--password` — пароль, по умолчанию — пустая строка.
- `--query`, `-q` — запрос для выполнения, при использовании в неинтерактивном режиме.
- `--queries-file`, `-qf` — путь к файлу с запросами для выполнения. Необходимо указать только одну из опций: `query` или `queries-file`.
- `--database`, `-d` — выбрать текущую БД. Без указания значение берется из настроек сервера (по умолчанию — БД `'default'`).
- `--multiline`, `-m` — если указано — разрешить многострочные запросы, не отправлять запрос по нажатию `Enter`.
- `--multiquery`, `-n` — если указано — разрешить выполнять несколько запросов, разделённых точкой с запятой.

- `--format`, `-f` — использовать указанный формат по умолчанию для вывода результата.
- `--vertical`, `-E` — если указано, использовать по умолчанию формат **Vertical** для вывода результата. То же самое, что `format=Vertical`. В этом формате каждое значение выводится на отдельной строке, что удобно для отображения широких таблиц.
- `--time`, `-t` — если указано, в неинтерактивном режиме вывести время выполнения запроса в поток `'stderr'`.
- `--stacktrace` — если указано, в случае исключения, выводить также его стек-трейс.
- `--config-file` — имя конфигурационного файла.
- `--secure` — если указано, будет использован безопасный канал.
- `--history_file` - путь к файлу с историей команд.
- `--param_<name>` — значение параметра для [запроса с параметрами](#).

Начиная с версии 20.5, в `clickhouse-client` есть автоматическая подсветка синтаксиса (включена всегда).

## Конфигурационные файлы

`clickhouse-client` использует первый существующий файл из:

- Определенного параметром `--config-file`.
- `./clickhouse-client.xml`
- `~/.clickhouse-client/config.xml`
- `/etc/clickhouse-client/config.xml`

Пример конфигурационного файла:

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

## Формат ID запроса

В интерактивном режиме `clickhouse-client` показывает ID для каждого запроса. По умолчанию ID выводится в таком виде:

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

Произвольный формат ID можно задать в конфигурационном файле внутри тега `query_id_formats`. ID подставляется вместо `{query_id}` в строке формата. В теге может быть перечислено несколько строк формата.

Эта возможность может быть полезна для генерации URL, с помощью которых выполняется профилирование запросов.

### Пример

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

Если применить приведённую выше конфигурацию, то ID запроса будет выводиться в следующем виде:

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

## Родной интерфейс (TCP)

Нативный протокол используется в [клиенте командной строки](#), для взаимодействия между серверами во время обработки распределенных запросов, а также в других программах на C++. К сожалению, у родного протокола ClickHouse пока нет формальной спецификации, но в нем можно разобраться с использованием исходного кода ClickHouse (начиная с [примерно этого места](#)) и/или путем перехвата и анализа TCP трафика.

## HTTP-интерфейс

HTTP интерфейс позволяет использовать ClickHouse на любой платформе, из любого языка программирования. У нас он используется для работы из Java и Perl, а также из shell-скриптов. В других отделах HTTP интерфейс используется из Perl, Python и Go. HTTP интерфейс более ограничен по сравнению с родным интерфейсом, но является более совместимым.

По умолчанию `clickhouse-server` слушает HTTP на порту 8123 (это можно изменить в конфиге). Если запросить `GET /` без параметров, то вернётся строка заданная с помощью настройки `http_server_default_response`. Значение по умолчанию «Ok.» (с переводом строки на конце).

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

Веб-интерфейс доступен по адресу: <http://localhost:8123/play>.

В скриптах проверки доступности вы можете использовать `GET /ping` без параметров. Если сервер доступен, всегда возвращается «Ok.» (с переводом строки на конце).

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

Запрос отправляется в виде URL параметра с именем `query`. Или как тело запроса при использовании метода POST.

Или начало запроса в URL параметре `query`, а продолжение POST-ом (зачем это нужно, будет объяснено ниже). Размер URL ограничен 16KB, это следует учитывать при отправке больших запросов.

В случае успеха возвращается код ответа 200 и результат обработки запроса в теле ответа, в случае ошибки — код ответа 500 и текст с описанием ошибки в теле ответа.

При использовании метода GET выставляется настройка `readonly`. То есть, для запросов, модифицирующих данные, можно использовать только метод POST. Сам запрос при этом можно отправлять как в теле POST запроса, так и в параметре URL.

Примеры:

```
$ 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
```

Как видно, curl немного неудобен тем, что надо URL-эскейпить пробелы.

Хотя wget сам всё эскейпит, но его не рекомендуется использовать, так как он плохо работает по HTTP 1.1 при использовании `keep-alive` и `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
```

Если часть запроса отправляется в параметре, а часть POST запросом, то между этими двумя кусками данных ставится перевод строки.

Пример (так работать не будет):

```
$ 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
```

По умолчанию данные возвращаются в формате `TabSeparated`.

Можно указать любой другой формат с помощью секции `FORMAT` запроса.

Кроме того, вы можете использовать параметр URL-адреса `default_format` или заголовок `X-ClickHouse-Format`, чтобы указать формат по умолчанию, отличный от `TabSeparated`.

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

1	
1	

Возможность передавать данные с помощью POST нужна для запросов `INSERT`. В этом случае вы можете написать начало запроса в параметре URL, а вставляемые данные передать POST запросом. Вставляемыми данными может быть, например, tab-separated дамп, полученный из MySQL. Таким образом, запрос `INSERT` заменяет `LOAD DATA LOCAL INFILE` из MySQL.

## Примеры

Создаём таблицу:

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

Используем привычный запрос `INSERT` для вставки данных:

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

Данные можно отправить отдельно от запроса:

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

Можно указать любой формат для данных. Формат Values - то же, что используется при записи `INSERT INTO t VALUES`:

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

Можно вставить данные из tab-separated дампа, указав соответствующий формат:

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

Прочитаем содержимое таблицы. Данные выводятся в произвольном порядке из-за параллельной обработки запроса:

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

Удаляем таблицу.

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

Для запросов, которые не возвращают таблицу с данными, в случае успеха выдаётся пустое тело ответа.

## Сжатие

Сжатие можно использовать для уменьшения трафика по сети при передаче большого количества данных, а также для создания сразу сжатых дампов.

Вы можете использовать внутренний формат сжатия Clickhouse при передаче данных. Формат сжатых данных нестандартный, и вам придётся использовать для работы с ним специальную программу `clickhouse-compressor`. Она устанавливается вместе с пакетом `clickhouse-client`. Для повышения эффективности вставки данных можно отключить проверку контрольной суммы на стороне сервера с помощью настройки

`http_native_compression_disable_checksumming_on_decompress`.

Если вы указали `compress=1` в URL, то сервер сжимает данные, которые он отправляет. Если вы указали `decompress=1` в URL, сервер распаковывает те данные, которые вы передаёте методом POST.

Также можно использовать [сжатие HTTP](#). ClickHouse поддерживает следующие [методы сжатия](#):

- `gzip`
- `br`
- `deflate`
- `xz`

Для отправки сжатого запроса POST добавьте заголовок `Content-Encoding: compression_method`.

Чтобы ClickHouse сжимал ответ, разрешите сжатие настройкой `enable_http_compression` и добавьте заголовок `Accept-Encoding: compression_method`. Уровень сжатия данных для всех методов сжатия можно задать с помощью настройки `http_zlib_compression_level`.

## Примечание

Некоторые HTTP-клиенты могут по умолчанию распаковывать данные (`gzip` и `deflate`) с сервера в фоновом режиме и вы можете получить распакованные данные, даже если правильно используете настройки сжатия.

## Примеры

```
## Отправка сжатых данных на сервер
$ echo "SELECT 1" | gzip -c | \
curl -sS --data-binary @- -H 'Content-Encoding: gzip' 'http://localhost:8123/'
```

```
## Получение сжатых данных с сервера
$ 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
```

## База данных по умолчанию

Вы можете использовать параметр URL `database` или заголовок `X-ClickHouse-Database`, чтобы указать БД по умолчанию.

```
$ 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
```

По умолчанию используется БД, которая прописана в настройках сервера, как БД по умолчанию. По умолчанию, это - БД default. Также вы всегда можете указать БД через точку перед именем таблицы.

Имя пользователя и пароль могут быть указаны в одном из трёх вариантов:

1. С использованием HTTP Basic Authentication. Пример:

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

1. В параметрах URL user и password. Пример:

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

1. С использованием заголовков 'X-ClickHouse-User' и 'X-ClickHouse-Key'. Пример:

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

Если пользователь не задан, то используется default. Если пароль не задан, то используется пустой пароль.

Также в параметрах URL вы можете указать любые настройки, которые будут использованы для обработки одного запроса, или целые профили настроек. Пример: [http://localhost:8123/?profile=web&max\\_rows\\_to\\_read=1000000000&query=SELECT+1](http://localhost:8123/?profile=web&max_rows_to_read=1000000000&query=SELECT+1)

Подробнее смотрите в разделе [Настройки](#).

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

Об остальных параметрах смотри раздел «[SET](#)».

Аналогично можно использовать ClickHouse-сессии в HTTP-протоколе. Для этого необходимо добавить к запросу GET параметр `session_id`. В качестве идентификатора сессии можно использовать произвольную строку. По умолчанию через 60 секунд бездействия сессия будет прервана. Можно изменить этот таймаут, изменения настройку `default_session_timeout` в конфигурации сервера, или добавив к запросу GET параметр `session_timeout`. Статус сессии можно проверить с помощью параметра `session_check=1`. В рамках одной сессии одновременно может исполняться только один запрос.

Прогресс выполнения запроса можно отслеживать с помощью заголовков ответа X-ClickHouse-Progress. Для этого включите [send\\_progress\\_in\\_http\\_headers](#). Пример последовательности заголовков:

```
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"}
```

Возможные поля заголовка:

- `read_rows` — количество прочитанных строк.
- `read_bytes` — объём прочитанных данных в байтах.
- `total_rows_to_read` — общее количество строк для чтения.
- `written_rows` — количество записанных строк.
- `written_bytes` — объём записанных данных в байтах.

Запущенные запросы не останавливаются автоматически при разрыве HTTP соединения. Парсинг и форматирование данных производится на стороне сервера и использование сети может быть неэффективным.

Может быть передан необязательный параметр `query_id` - идентификатор запроса, произвольная строка. Подробнее смотрите раздел «Настройки, `replace_running_query`».

Может быть передан необязательный параметр `quota_key` - ключ квоты, произвольная строка. Подробнее смотрите раздел «Квоты».

HTTP интерфейс позволяет передать внешние данные (внешние временные таблицы) для использования запроса. Подробнее смотрите раздел «Внешние данные для обработки запроса»

## Буферизация ответа

Существует возможность включить буферизацию ответа на стороне сервера. Для этого предусмотрены параметры URL `buffer_size` и `wait_end_of_query`.

`buffer_size` определяет количество байт результата которые будут буферизованы в памяти сервера. Если тело результата больше этого порога, то буфер будет переписан в HTTP канал, а оставшиеся данные будут отправляться в HTTP-канал напрямую.

Чтобы гарантировать буферизацию всего ответа, необходимо выставить `wait_end_of_query=1`. В этом случае данные, не поместившиеся в памяти, будут буферизованы во временном файле сервера.

Пример:

```
$ 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'
```

Буферизация позволяет избежать ситуации, когда код ответа и HTTP-заголовки были отправлены клиенту, после чего возникла ошибка выполнения запроса. В такой ситуации сообщение об ошибке записывается в конце тела ответа, и на стороне клиента ошибка может быть обнаружена только на этапе парсинга.

## Запросы с параметрами

Можно создать запрос с параметрами и передать для них значения из соответствующих параметров HTTP-запроса. Дополнительную информацию смотрите в [Запросы с параметрами для консольного клиента](#).

## Пример

```
$ curl -sS "http://localhost:8123/?param_id=2&param_phrase=test" -d "SELECT * FROM table WHERE int_column = {id:UInt8} and string_column = {phrase:String}"
```

## Предопределенный HTTP интерфейс

ClickHouse поддерживает определенные запросы через HTTP-интерфейс. Например, вы можете записать данные в таблицу следующим образом:

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

ClickHouse также поддерживает предопределенный HTTP-интерфейс, который может помочь вам легче интегрироваться со сторонними инструментами, такими как [Prometheus exporter](#).

Пример:

- Прежде всего, добавьте раздел в конфигурационный файл сервера:

```
<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>
```

- Теперь вы можете напрямую запросить URL-адрес для получения данных в формате Prometheus:

```

$ 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

```

Как вы можете видеть из примера, `http_handlers` настраивается в файле `config.xml` и может содержать несколько правил. ClickHouse будет сопоставлять полученные HTTP-запросы с предопределенным типом в правиле, и первое совпадение запустит обработчик. Затем ClickHouse выполнит соответствующий предопределенный запрос.

В настоящий момент с помощью `rule` можно настроить `method`, `headers`, `url`, `handler`:

- `method` отвечает за соответствие метода HTTP-запроса. `method` соответствует методу `method` протокола HTTP. Это необязательная настройка. Если она не определена в файле конфигурации, она не соответствует методу HTTP-запроса.

- `url` отвечает за соответствие URL HTTP-запроса. Она совместима с регулярными выражениями `RE2`. Это необязательная настройка. Если она не определена в файле конфигурации, она не соответствует URL-адресу HTTP-запроса.
- `headers` отвечают за соответствие заголовка HTTP-запроса. Она совместима с регулярными выражениями `RE2`. Это необязательная настройка. Если она не определена в файле конфигурации, она не соответствует заголовку HTTP-запроса.

- handler содержит основную часть обработчика. Сейчас handler может настраивать type, status, content\_type, response\_content, query, query\_param\_name.
- type на данный момент поддерживает три типа: **predefined\_query\_handler**, **dynamic\_query\_handler**, **static**.
- query — используется с типом predefined\_query\_handler, выполняет запрос при вызове обработчика.
  - query\_param\_name — используется с типом dynamic\_query\_handler, извлекает и выполняет значение, соответствующее значению query\_param\_name в параметрах HTTP-запроса.
  - status — используется с типом static, возвращает код состояния ответа.
  - content\_type — используется с типом static, возвращает content-type.
  - response\_content — используется с типом static, содержимое ответа, отправленное клиенту, при использовании префикса 'file://' or 'config://', находит содержимое из файла или конфигурации, отправленного клиенту.

Далее приведены методы настройки для различных типов.

## predefined\_query\_handler

predefined\_query\_handler поддерживает настройки Settings и query\_params значений. Вы можете настроить запрос в типе predefined\_query\_handler.

Значение query — это предопределенный запрос predefined\_query\_handler, который выполняется ClickHouse при совпадении HTTP-запроса и возврате результата запроса. Это обязательная настройка.

В следующем примере определяются настройки max\_threads и max\_alter\_threads, а затем запрашивается системная таблица, чтобы проверить, были ли эти параметры успешно установлены.

Пример:

```
<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
```

## Предупреждение

В одном predefined\_query\_handler поддерживается только один запрос типа INSERT.

## dynamic\_query\_handler

В `dynamic_query_handler`, запрос пишется в виде параметров HTTP-запроса. Разница в том, что в `predefined_query_handler`, запрос записывается в конфигурационный файл. Вы можете настроить `query_param_name` в `dynamic_query_handler`.

ClickHouse извлекает и выполняет значение, соответствующее значению `query_param_name` URL-адресе HTTP-запроса. Значение по умолчанию `query_param_name` — это `/query`. Это необязательная настройка. Если в файле конфигурации нет определения, параметр не передается.

Чтобы поэкспериментировать с этой функциональностью, в примере определяются значения `max_threads` и `max_alter_threads` и запрашивается, успешно ли были установлены настройки.

Пример:

```
<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` может возвращать `content_type`, `status` и `response_content`. `response_content` может возвращать конкретное содержимое.

Пример:

Возвращает сообщение.

```
<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!%
```

Находит содержимое настроек отправленных клиенту.

```
<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>%
```

Находит содержимое файла, отправленного клиенту.

```

<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-интерфейс

ClickHouse поддерживает взаимодействие по протоколу MySQL. Данная функция включается настройкой `mysql_port` в конфигурационном файле:

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

Пример подключения с помощью стандартного клиента mysql:

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

Вывод в случае успешного подключения:

```
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>
```

Для совместимости со всеми клиентами рекомендуется задавать пароль пользователя в конфигурационном файле с помощью двойного хэша `SHA1`.

В случае указания пароля с помощью `SHA256` некоторые клиенты не смогут пройти аутентификацию (mysqljs и старые версии стандартного клиента mysql).

Ограничения:

- не поддерживаются подготовленные запросы
- некоторые типы данных отправляются как строки

Чтобы прервать долго выполняемый запрос, используйте запрос `KILL QUERY connection_id` (во время выполнения он будет заменен на `KILL QUERY WHERE query_id = connection_id`). Например:

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

## Форматы входных и выходных данных

ClickHouse может принимать (`INSERT`) и отдавать (`SELECT`) данные в различных форматах.

Поддерживаемые форматы и возможность использовать их в запросах `INSERT` и `SELECT` перечислены в таблице ниже.

Формат	INSERT	SELECT
TabSeparated	✓	✓
TabSeparatedRaw	✓	✓
TabSeparatedWithNames	✓	✓

Формат	INSERT	SELECT
TabSeparatedWithNamesAndTypes	✓	✓
Template	✓	✓
TemplateIgnoreSpaces	✓	✗
CSV	✓	✓
CSVWithNames	✓	✓
CustomSeparated	✓	✓
Values	✓	✓
Vertical	✗	✓
JSON	✗	✓
JSONAsString	✓	✗
JSONStrings	✗	✓
JSONCompact	✗	✓
JSONCompactStrings	✗	✓
JSONEachRow	✓	✓
JSONEachRowWithProgress	✗	✓
JSONStringsEachRow	✓	✓
JSONStringsEachRowWithProgress	✗	✓
JSONCompactEachRow	✓	✓
JSONCompactEachRowWithNamesAndTypes	✓	✓
JSONCompactStringsEachRow	✓	✓
JSONCompactStringsEachRowWithNamesAndTypes	✓	✓
TSKV	✓	✓
Pretty	✗	✓
PrettyCompact	✗	✓
PrettyCompactMonoBlock	✗	✓

Формат	INSERT	SELECT
PrettyNoEscapes	✗	✓
PrettySpace	✗	✓
Protobuf	✓	✓
ProtobufSingle	✓	✓
Avro	✓	✓
AvroConfluent	✓	✗
Parquet	✓	✓
Arrow	✓	✓
ArrowStream	✓	✓
ORC	✓	✓
RowBinary	✓	✓
RowBinaryWithNamesAndTypes	✓	✓
Native	✓	✓
Null	✗	✓
XML	✗	✓
CapnProto	✓	✗
LineAsString	✓	✗
Regexp	✓	✗
RawBLOB	✓	✓
MsgPack	✓	✓

Вы можете регулировать некоторые параметры работы с форматами с помощью настроек ClickHouse. За дополнительной информацией обращайтесь к разделу [Настройки](#).

## TabSeparated

В TabSeparated формате данные пишутся по строкам. Каждая строчка содержит значения, разделённые табами. После каждого значения идёт таб, кроме последнего значения в строке, после которого идёт перевод строки. Везде подразумеваются исключительно unix-переводы строк. Последняя строка также обязана содержать перевод строки на конце. Значения пишутся в текстовом виде, без обрамляющих кавычек, с экранированием служебных символов.

Этот формат также доступен под именем TSV.

Формат TabSeparated удобен для обработки данных произвольными программами и скриптами. Он используется по умолчанию в HTTP-интерфейсе, а также в batch-режиме клиента командной строки. Также формат позволяет переносить данные между разными СУБД. Например, вы можете получить дамп из MySQL и загрузить его в ClickHouse, или наоборот.

Формат TabSeparated поддерживает вывод тотальных значений (при использовании WITH TOTALS) и экстремальных значений (при настройке extremes выставленной в 1). В этих случаях, после основных данных выводятся тотальные значения, и экстремальные значения. Основной результат, тотальные значения и экстремальные значения, отделяются друг от друга пустой строкой. Пример:

```
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

## Форматирование данных

Целые числа пишутся в десятичной форме. Числа могут содержать лишний символ «+» в начале (игнорируется при парсинге, а при форматировании не пишется). Неотрицательные числа не могут содержать знак отрицания. При чтении допустим парсинг пустой строки, как числа ноль, или (для знаковых типов) строки, состоящей из одного минуса, как числа ноль. Числа, не помещающиеся в соответствующий тип данных, могут парситься, как некоторое другое число, без сообщения об ошибке.

Числа с плавающей запятой пишутся в десятичной форме. При этом, десятичный разделитель - точка. Поддерживается экспоненциальная запись, а также inf, +inf, -inf, nan. Запись числа с плавающей запятой может начинаться или заканчиваться на десятичную точку.  
При форматировании возможна потеря точности чисел с плавающей запятой.  
При парсинге, допустимо чтение не обязательно наиболее близкого к десятичной записи машинно-представимого числа.

Даты выводятся в формате YYYY-MM-DD, парсятся в том же формате, но с любыми символами в качестве разделителей.

Даты-с-временем выводятся в формате YYYY-MM-DD hh:mm:ss, парсятся в том же формате, но с любыми символами в качестве разделителей.

Всё это происходит в системном часовом поясе на момент старта клиента (если клиент занимается форматированием данных) или сервера. Для дат-с-временем не указывается, действует ли daylight saving time. То есть, если в дампе есть времена во время перевода стрелок назад, то дамп не соответствует данным однозначно, и при парсинге будет выбрано какое-либо из двух времён.  
При парсинге, некорректные даты и даты-с-временем могут парситься с естественным переполнением или как нулевые даты/даты-с-временем без сообщения об ошибке.

В качестве исключения, поддерживается также парсинг даты-с-временем в формате unix timestamp, если он состоит ровно из 10 десятичных цифр. Результат не зависит от часового пояса. Различие форматов YYYY-MM-DD hh:mm:ss и NNNNNNNNN делается автоматически.

Строки выводятся с экранированием спецсимволов с помощью обратного слеша. При выводе, используются следующие escape-последовательности: \b, \f, \r, \n, \t, \0, \v, \N. Парсер также поддерживает последовательности \a, \v, и \xHH (последовательности hex escape) и любые последовательности вида \c, где c — любой символ (такие последовательности преобразуются в c). Таким образом, при чтении поддерживаются форматы, где перевод строки может быть записан как \n и как \ и перевод строки. Например, строка Hello world, где между словами вместо пробела стоит перевод строки, может быть считана в любом из следующих вариантов:

```
Hello\nworld
```

```
Hello\
world
```

Второй вариант поддерживается, так как его использует MySQL при записи tab-separated дампа.

Минимальный набор символов, которых вам необходимо экранировать при передаче в TabSeparated формате: таб, перевод строки (LF) и обратный слеш.

Экранируется лишь небольшой набор символов. Вы можете легко наткнуться на строковое значение, которое испортит ваш терминал при выводе в него.

Массивы форматируются в виде списка значений через запятую в квадратных скобках. Элементы массива - числа форматируются как обычно, а даты, даты-с-временем и строки - в одинарных кавычках с такими же правилами экранирования, как указано выше.

**NULL** форматируется как \N.

Каждый элемент структуры типа **Nested** представляется как отдельный массив.

Например:

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

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

```
SELECT * FROM nesteddt FORMAT TSV
```

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

## TabSeparatedRaw

Отличается от формата TabSeparated тем, что строки выводятся без экранирования.

Используя этот формат, следите, чтобы в полях не было символов табуляции или разрыва строки.

Этот формат также доступен под именем TSVRaw.

## TabSeparatedWithNames

Отличается от формата TabSeparated тем, что в первой строке пишутся имена столбцов. При парсинге, первая строка полностью игнорируется. Вы не можете использовать имена столбцов, чтобы указать их порядок расположения, или чтобы проверить их корректность. (Поддержка обработки заголовка при парсинге может быть добавлена в будущем.)

Этот формат также доступен под именем TSVWithNames.

## TabSeparatedWithNamesAndTypes

Отличается от формата TabSeparated тем, что в первой строке пишутся имена столбцов, а во второй - типы столбцов.

При парсинге, первая и вторая строка полностью игнорируются.

Этот формат также доступен под именем TSVWithNamesAndTypes.

## Template

Этот формат позволяет указать произвольную форматную строку, в которую подставляются значения, сериализованные выбранным способом.

Для этого используются настройки format\_template\_resultset, format\_template\_row, format\_template\_rows\_between\_delimiter и настройки экранирования других форматов (например, output\_format\_json\_quote\_64bit\_integers при экранировании как в JSON, см. далее)

Настройка format\_template\_row задаёт путь к файлу, содержащему форматную строку для строк таблицы, которая должна иметь вид:

```
delimiter_1${column_1:serializeAs_1}delimiter_2${column_2:serializeAs_2} ... delimiter_N,
```

где `delimiter\_i` - разделители между значениями (символ `\$` в разделителе экранируется как `\$\$`), `column\_i` - имена или номера столбцов, значения которых должны быть выведены или считаны (если имя не указано - столбец пропускается), `serializeAs\_i` - тип экранирования для значений соответствующего столбца. Поддерживаются следующие типы экранирования:

- `CSV`, `JSON`, `XML` (как в одноимённых форматах)
- `Escaped` (как в `TSV`)
- `Quoted` (как в `Values`)
- `Raw` (без экранирования, как в `TSVRaw`)
- `None` (тип экранирования отсутствует, см. далее)

Если для столбца не указан тип экранирования, используется `None`. `XML` и `Raw` поддерживаются только для вывода.

Так, в форматной строке

```
`Search phrase: ${SearchPhrase:Quoted}, count: ${c:Escaped}, ad price: $$ ${price:JSON};`
```

между разделителями `Search phrase: `, `count: `, `ad price: \$` и `;` при выводе будут подставлены (при вводе - будут ожидаться) значения столбцов `SearchPhrase`, `c` и `price`, сериализованные как `Quoted`, `Escaped` и `JSON` соответственно, например:

```
`Search phrase: 'bathroom interior design', count: 2166, ad price: $3;`
```

Настройка format\_template\_rows\_between\_delimiter задаёт разделитель между строками, который выводится (или ожидается при вводе) после каждой строки, кроме последней. По умолчанию `\n`.

Настройка format\_template\_resultset задаёт путь к файлу, содержащему форматную строку для результата. Форматная строка для результата имеет синтаксис аналогичный форматной строке для строк таблицы и позволяет указать префикс, суффикс и способ вывода дополнительной информации. Вместо имён столбцов в ней указываются следующие имена подстановок:

- `data` - строки с данными в формате `format_template_row`, разделённые `format_template_rows_between_delimiter`. Эта подстановка должна быть первой подстановкой в форматной строке.
- `totals` - строка с тотальными значениями в формате `format_template_row` (при использовании `WITH TOTALS`)
- `min` - строка с минимальными значениями в формате `format_template_row` (при настройке `extremes`, выставленной в 1)
- `max` - строка с максимальными значениями в формате `format_template_row` (при настройке `extremes`, выставленной в 1)
- `rows` - общее количество выведенных строчек
- `rows_before_limit` - не менее скольких строчек получилось бы, если бы не было `LIMIT`-а. Выводится только если запрос содержит `LIMIT`. В случае, если запрос содержит `GROUP BY`, `rows_before_limit` - точное число строк, которое получилось бы, если бы не было `LIMIT`-а.
- `time` - время выполнения запроса в секундах
- `rows_read` - сколько строк было прочитано при выполнении запроса
- `bytes_read` - сколько байт (несжатых) было прочитано при выполнении запроса

У подстановок `data`, `totals`, `min` и `max` не должны быть указаны типы экранирования (или должен быть указан `None`). Остальные подстановки - это отдельные значения, для них может быть указан любой тип экранирования.

Если строка `format_template_resultset` пустая, то по-умолчанию используется  `${data}`.

Из всех перечисленных подстановок форматная строка `format_template_resultset` для ввода может содержать только `data`.

Также при вводе формат поддерживает пропуск значений столбцов и пропуск значений в префикссе и суффиксе (см. пример).

Пример вывода:

```
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>
```

Результат:

```
<!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>
```

Пример ввода:

```
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 и Sign внутри подстановок - имена столбцов в таблице, в которую вставляются данные. Значения после Useless field в строках и значение после \nTotal rows: в суффиксе будут проигнорированы.

Все разделители во входных данных должны строго соответствовать разделителям в форматных строках.

## TemplateIgnoreSpaces

Подходит только для ввода. Отличается от формата `Template` тем, что пропускает пробельные символы между разделителями и значениями во входном потоке. Также в этом формате можно указать пустые подстановки с типом экранирования `None` (`${}` или `${:None}`), чтобы разбить разделители на несколько частей, пробелы между которыми должны игнорироваться. Такие подстановки используются только для пропуска пробелов. С помощью этого формата можно считывать `JSON`, если значения столбцов в нём всегда идут в одном порядке в каждой строке. Например, для вставки данных из примера вывода формата `JSON` в таблицу со столбцами `phrase` и `cnt` можно использовать следующий запрос:

```
INSERT INTO table_name FORMAT TemplateIgnoreSpaces SETTINGS
format_schema = '${"meta":{}:$:{JSON},${"data":{}:$:{}
[$data]}${},${"totals":{}:$:{JSON}},${"extremes":{}:$:{JSON}},${"rows":{}:$:{JSON}},${"rows_before_limit_
at_least":{}:$:{JSON}}${}',
format_schema_rows = '${"SearchPhrase":{}:$:{phrase:JSON}[],${"c":{}:$:{}$:{cnt:JSON}[]}}',
format_schema_rows_between_delimiter = ','
```

## TSKV

Похож на TabSeparated, но выводит значения в формате name=value. Имена экранируются так же, как строки в формате TabSeparated и, дополнительно, экранируется также символ =.

```
SearchPhrase= count()=8267016
SearchPhrase=интерьер ванной комнаты count()=2166
SearchPhrase=яндекс count()=1655
SearchPhrase=весна 2014 мода count()=1549
SearchPhrase=фриформ фото count()=1480
SearchPhrase=анджелина джоли count()=1245
SearchPhrase=омск count()=1112
SearchPhrase=фото собак разных пород count()=1091
SearchPhrase=дизайн штор count()=1064
SearchPhrase=баку count()=1000
```

NULL форматируется как \N.

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

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

При большом количестве маленьких столбцов, этот формат существенно неэффективен, и обычно нет причин его использовать. Впрочем, он не хуже формата JSONEachRow по производительности.

Поддерживается как вывод, так и парсинг данных в этом формате. При парсинге, поддерживается расположение значений разных столбцов в произвольном порядке. Допустимо отсутствие некоторых значений - тогда они воспринимаются как равные значениям по умолчанию. В этом случае в качестве значений по умолчанию используются нули и пустые строки. Сложные значения, которые могут быть заданы в таблице не поддерживаются как значения по умолчанию.

При парсинге, в качестве дополнительного поля, может присутствовать `tskv` без знака равенства и без значения. Это поле игнорируется.

## CSV

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

При форматировании, строки выводятся в двойных кавычках. Двойная кавычка внутри строки выводится как две двойные кавычки подряд. Других правил экранирования нет. Даты и даты-с-временем выводятся в двойных кавычках. Числа выводятся без кавычек. Значения разделяются символом-разделителем, по умолчанию — ,. Символ-разделитель определяется настройкой `format_csv_delimiter`. Строки разделяются unix переводом строки (LF). Массивы сериализуются в CSV следующим образом: сначала массив сериализуется в строку, как в формате TabSeparated, а затем полученная строка выводится в CSV в двойных кавычках. Кортежи в формате CSV сериализуются, как отдельные столбцы (то есть, теряется их вложенность в кортеж).

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

\*По умолчанию — ,. См. настройку `format_csv_delimiter` для дополнительной информации.

При парсинге, все значения могут парситься как в кавычках, так и без кавычек. Поддерживаются как двойные, так и одинарные кавычки. Строки также могут быть без кавычек. В этом случае они парсятся до символа-разделителя или перевода строки (CR или LF). В нарушение RFC, в случае парсинга строк не в кавычках, начальные и конечные пробелы и табы игнорируются. В качестве перевода строки, поддерживаются как Unix (LF), так и Windows (CR LF) и Mac OS Classic (LF CR) варианты.

`NULL` форматируется в виде `\N` или `NULL` или пустой неэкранированной строки (см. настройки `input_format_csv_unquoted_null_literal_as_null` и `input_format_defaults_for_omitted_fields`).

Если установлена настройка `input_format_defaults_for_omitted_fields = 1` и тип столбца не `Nullable(T)`, то пустые значения без кавычек заменяются значениями по умолчанию для типа данных столбца.

Формат CSV поддерживает вывод `totals` и `extremes` аналогично `TabSeparated`.

## CSVWithNames

Выводит также заголовок, аналогично `TabSeparatedWithNames`.

## CustomSeparated

Аналогичен `Template`, но выводит (или считывает) все столбцы, используя для них правило экранирования из настройки `format_customEscapingRule` и разделители из настроек `formatCustomFieldDelimiter`, `formatCustomRowBeforeDelimiter`, `formatCustomRowAfterDelimiter`, `formatCustomRowBetweenDelimiter`, `formatCustomResultBeforeDelimiter` и `formatCustomResultAfterDelimiter`, а не из форматных строк.

Также существует формат `CustomSeparatedIgnoreSpaces`, аналогичный `TemplateIgnoreSpaces`.

## JSON

Выводит данные в формате JSON. Кроме таблицы с данными, также выводятся имена и типы столбцов, и некоторая дополнительная информация - общее количество выведенных строк, а также количество строк, которое могло бы быть выведено, если бы не было `LIMIT`-а. Пример:

```
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
}
```

JSON совместим с JavaScript. Для этого, дополнительно экранируются некоторые символы: символ прямого слеша / экранируется в виде \; альтернативные переводы строк U+2028, U+2029, на которых ломаются некоторые браузеры, экранируются в виде \uXXXX-последовательностей. Экранируются ASCII control characters: backspace, form feed, line feed, carriage return, horizontal tab в виде \b, \f, \n, \r, \t соответственно, а также остальные байты из диапазона 00-1F с помощью \uXXXX-последовательностей. Невалидные UTF-8 последовательности заменяются на replacement character ♦ и, таким образом, выводимый текст будет состоять из валидных UTF-8 последовательностей. Числа типа UInt64 и Int64, для совместимости с JavaScript, по умолчанию выводятся в двойных кавычках. Чтобы они выводились без кавычек, можно установить конфигурационный параметр [output\\_format\\_json\\_quote\\_64bit\\_integers](#) равным 0.

`rows` - общее количество выведенных строчек.

`rows_before_limit_at_least` - не менее скольких строчек получилось бы, если бы не было LIMIT-а.

Выводится только если запрос содержит LIMIT.

В случае, если запрос содержит GROUP BY, `rows_before_limit_at_least` - точное число строк, которое получилось бы, если бы не было LIMIT-а.

`totals` - тотальные значения (при использовании WITH TOTALS).

`extremes` - экстремальные значения (при настройке extremes, выставленной в 1).

Этот формат подходит только для вывода результата выполнения запроса, но не для парсинга (приёма данных для вставки в таблицу).

ClickHouse поддерживает **NULL**, который при выводе JSON будет отображен как `null`. Чтобы включить отображение в результате значений `+nan`, `-nan`, `+inf`, `-inf`, установите параметр `output_format_json_quote_denormals` равным 1.

## Смотрите также

- Формат [JSONEachRow](#)
- Настройка `output_format_json_array_of_rows`

## JSONStrings

Отличается от JSON только тем, что поля данных выводятся в строках, а не в типизированных значениях JSON.

Пример:

```
{  
    "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

В этом формате один объект JSON интерпретируется как одно строковое значение. Если входные данные имеют несколько объектов JSON, разделенных запятой, то они интерпретируются как отдельные строки таблицы. Если входные данные заключены в квадратные скобки, они интерпретируются как массив JSON-объектов.

В этом формате парситься может только таблица с единственным полем типа **String**. Остальные столбцы должны быть заданы как **DEFAULT** или **MATERIALIZED**(смотрите раздел [Значения по умолчанию](#)), либо отсутствовать. Для дальнейшей обработки объекта JSON, представленного в строке, вы можете использовать [функции для работы с JSON](#).

## Пример

Запрос:

```
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;
```

Результат:

```
json
{"foo": {"bar": {"x": "y"}, "baz": 1}} |
{} |
{"any json stucture": 1} |
```

## Пример с массивом объектов JSON

Запрос:

```
DROP TABLE IF EXISTS json_square_brackets;
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;
```

Результат:

```
field
{"id": 1, "name": "name1"} |
{"id": 2, "name": "name2"} |
```

## JSONCompact

### JSONCompactStrings

Отличается от JSON только тем, что строчки данных выводятся в массивах, а не в object-ах.

Пример:

```
//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

При использовании этих форматов ClickHouse выводит каждую запись как значения JSON (каждое значение отдельной строкой), при этом данные в целом — невалидный JSON.

```
{"some_int":42,"some_str":"hello","some_tuple":[1,"a"]} // JSONEachRow  
[42,"hello",[1,"a"]] // JSONCompactEachRow  
["42","hello","(2,'a')"] // JSONCompactStringsEachRow
```

При вставке данных вы должны предоставить отдельное значение JSON для каждой строки.

## JSONEachRowWithProgress

## JSONStringsEachRowWithProgress

Отличается от JSONEachRow/JSONStringsEachRow тем, что ClickHouse будет выдавать информацию о ходе выполнения в виде значений JSON.

```
{"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

Отличается от JSONCompactEachRow/JSONCompactStringsEachRow тем, что имена и типы столбцов записываются как первые две строки.

```
["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]]
```

## Вставка данных

```
INSERT INTO UserActivity FORMAT JSONEachRow {"PageViews":5, "UserID":"4324182021466249494",  
"Duration":146,"Sign": -1} {"UserID":"4324182021466249494", "PageViews":6, "Duration":185, "Sign": 1}
```

ClickHouse допускает:

- Любой порядок пар ключ-значение в объекте.
- Пропуск отдельных значений.

ClickHouse игнорирует пробелы между элементами и запятые после объектов. Вы можете передать все объекты одной строкой. Вам не нужно разделять их переносами строк.

## Обработка пропущенных значений

ClickHouse заменяет опущенные значения значениями по умолчанию для соответствующих [data types](#).

Если указано `DEFAULT expr`, то ClickHouse использует различные правила подстановки в зависимости от настройки `input_format_defaults_for_omitted_fields`.

Рассмотрим следующую таблицу:

```
CREATE TABLE IF NOT EXISTS example_table
(
    x UInt32,
    a DEFAULT x * 2
) ENGINE = Memory;
```

- Если `input_format_defaults_for_omitted_fields` = 0, то значение по умолчанию для `x` и `a` равняется 0 (поскольку это значение по умолчанию для типа данных `UInt32`.)
- Если `input_format_defaults_for_omitted_fields` = 1, то значение по умолчанию для `x` равно 0, а значение по умолчанию `a` равно `x * 2`.

## Предупреждение

Если `input_format_defaults_for_omitted_fields` = 1, то при обработке запросов ClickHouse потребляет больше вычислительных ресурсов, чем если `input_format_defaults_for_omitted_fields` = 0.

## Выборка данных

Рассмотрим в качестве примера таблицу `UserActivity`:

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

Запрос `SELECT * FROM UserActivity FORMAT JSONEachRow` возвращает:

```
{"UserID":"4324182021466249494","PageViews":5,"Duration":146,"Sign":-1}
{"UserID":"4324182021466249494","PageViews":6,"Duration":185,"Sign":1}
```

В отличие от формата `JSON`, для `JSONEachRow` ClickHouse не заменяет невалидные UTF-8 последовательности. Значения экранируются так же, как и для формата `JSON`.

## Примечание

В строках может выводиться произвольный набор байт. Используйте формат `JSONEachRow`, если вы уверены, что данные в таблице могут быть представлены в формате `JSON` без потери информации.

## Использование вложенных структур

Если у вас есть таблица со столбцами типа `Nested`, то в неё можно вставить данные из `JSON`-документа с такой же структурой. Функциональность включается настройкой `input_format_import_nested_json`.

Например, рассмотрим следующую таблицу:

```
CREATE TABLE json_each_row_nested (n Nested (s String, i Int32) ) ENGINE = Memory
```

Из описания типа данных `Nested` видно, что ClickHouse трактует каждый компонент вложенной структуры как отдельный столбец (для нашей таблицы `n.s` и `n.i`). Можно вставить данные следующим образом:

```
INSERT INTO json_each_row_nested FORMAT JSONEachRow {"n.s": ["abc", "def"], "n.i": [1, 23]}
```

Чтобы вставить данные как иерархический объект JSON, установите [input\\_format\\_import\\_nested\\_json=1](#).

```
{  
  "n": {  
    "s": ["abc", "def"],  
    "i": [1, 23]  
  }  
}
```

Без этой настройки ClickHouse сгенерирует исключение.

```
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
```

n.s	n.i
['abc','def']	[1,23]

## Native

Самый эффективный формат. Данные пишутся и читаются блоками в бинарном виде. Для каждого блока пишется количество строк, количество столбцов, имена и типы столбцов, а затем кусочки столбцов этого блока, один за другим. То есть, этот формат является «столбцовым» - не преобразует столбцы в строки. Именно этот формат используется в родном интерфейсе - при межсерверном взаимодействии, при использовании клиента командной строки, при работе клиентов, написанных на C++.

Вы можете использовать этот формат для быстрой генерации дампов, которые могут быть прочитаны только СУБД ClickHouse. Вряд ли имеет смысл работать с этим форматом самостоятельно.

## Null

Ничего не выводит. При этом, запрос обрабатывается, а при использовании клиента командной строки, данные ещё и передаются на клиент. Используется для тестов, в том числе, тестов производительности.

Очевидно, формат подходит только для вывода, но не для парсинга.

## Pretty

Выводит данные в виде Unicode-art табличек, также используя ANSI-escape последовательности для установки цветов в терминале.

Рисуется полная сетка таблицы и, таким образом, каждая строчка занимает две строки в терминале.

Каждый блок результата выводится в виде отдельной таблицы. Это нужно, чтобы можно было выводить блоки без буферизации результата (буферизация потребовалась бы, чтобы заранее вычислить видимую ширину всех значений.)

`NULL` выводится как `\null`.

```
SELECT * FROM t_null
```

x	y
1	\null

В форматах Pretty\* строки выводятся без экранирования. Ниже приведен пример для формата `PrettyCompact`:

```
SELECT 'String with \'quotes\' and \t character' AS Escaping_test
```

```
Escaping_test
String with 'quotes' and\t character |
```

Для защиты от вываливания слишком большого количества данных в терминал, выводится только первые 10 000 строк. Если строк больше или равно 10 000, то будет написано «Showed first 10 000.» Этот формат подходит только для вывода результата выполнения запроса, но не для парсинга (приёма данных для вставки в таблицу).

Формат `Pretty` поддерживает вывод тотальных значений (при использовании `WITH TOTALS`) и экстремальных значений (при настройке `extremes` выставленной в 1). В этих случаях, после основных данных выводятся тотальные значения, и экстремальные значения, в отдельных табличках. Пример (показан для формата `PrettyCompact`):

```
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

Отличается от [Pretty](#) тем, что не рисуется сетка между строками - результат более компактный. Этот формат используется по умолчанию в клиенте командной строки в интерактивном режиме.

## PrettyCompactMonoBlock

Отличается от [PrettyCompact](#) тем, что строки (до 10 000 штук) буферизуются и затем выводятся в виде одной таблицы, а не по блокам.

## PrettyNoEscapes

Отличается от Pretty тем, что не используются ANSI-escape последовательности. Это нужно для отображения этого формата в браузере, а также при использовании утилиты командной строки `watch`.

Пример:

```
$ watch -n1 "clickhouse-client --query='SELECT event, value FROM system.events FORMAT PrettyCompactNoEscapes'"
```

Для отображения в браузере, вы можете использовать HTTP интерфейс.

## PrettyCompactNoEscapes

Аналогично.

## PrettySpaceNoEscapes

Аналогично.

## PrettySpace

Отличается от [PrettyCompact](#) тем, что вместо сетки используется пустое пространство (пробелы).

## RowBinary

Форматирует и парсит данные по строкам, в бинарном виде. Строки и значения уложены подряд, без разделителей.

Формат менее эффективен, чем формат Native, так как является строковым.

Числа представлены в little endian формате фиксированной длины. Для примера, UInt64 занимает 8 байт.

DateTime представлены как UInt32, содержащий unix timestamp в качестве значения.

Date представлены как UInt16, содержащий количество дней, прошедших с 1970-01-01 в качестве значения.

String представлены как длина в формате varint (unsigned LEB128), а затем байты строки.

FixedString представлены просто как последовательность байт.

Array представлены как длина в формате varint (unsigned LEB128), а затем элементы массива, подряд.

Для поддержки **NULL** перед каждым значением типа **Nullable** следует байт содержащий 1 или 0. Если байт 1, то значение равно NULL, и этот байт интерпретируется как отдельное значение (т.е. после него следует значение следующего поля). Если байт 0, то после байта следует значение поля (не равно NULL).

## RowBinaryWithNamesAndTypes

То же самое что **RowBinary**, но добавляется заголовок:

- Количество колонок - N, закодированное LEB128,
- N строк (String) с именами колонок,
- N строк (String) с типами колонок.

## Values

Выводит каждую строку в скобках. Строки разделены запятыми. После последней строки запятой нет. Значения внутри скобок также разделены запятыми. Числа выводятся в десятичном виде без кавычек. Массивы выводятся в квадратных скобках. Строки, даты, даты-с-временем выводятся в кавычках. Правила экранирования и особенности парсинга аналогичны формату **TabSeparated**. При формировании, лишние пробелы не ставятся, а при парсинге - допустимы и пропускаются (за исключением пробелов внутри значений типа массив, которые недопустимы). **NULL** представляется как **NULL**.

Минимальный набор символов, которых вам необходимо экранировать при передаче в Values формате: одинарная кавычка и обратный слеш.

Именно этот формат используется в запросе `INSERT INTO t VALUES ...`, но вы также можете использовать его для форматирования результатов запросов.

## Vertical

Выводит каждое значение на отдельной строке, с указанием имени столбца. Формат удобно использовать для вывода одной-нескольких строк, если каждая строка состоит из большого количества столбцов.

**NULL** выводится как **NULL**.

Пример:

```
SELECT * FROM t_null FORMAT Vertical
```

Row 1:

x: 1  
y: NULL

В формате Vertical строки выводятся без экранирования. Например:

```
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

Этот формат подходит только для вывода результата выполнения запроса, но не для парсинга (приёма данных для вставки в таблицу).

## XML

Формат XML подходит только для вывода данных, не для парсинга. Пример:

```

<?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>интерьер ванной комнаты</SearchPhrase>
            <field>2166</field>
        </row>
        <row>
            <SearchPhrase>яндекс</SearchPhrase>
            <field>1655</field>
        </row>
        <row>
            <SearchPhrase>весна 2014 мода</SearchPhrase>
            <field>1549</field>
        </row>
        <row>
            <SearchPhrase>фриформ фото</SearchPhrase>
            <field>1480</field>
        </row>
        <row>
            <SearchPhrase>анджелина джоли</SearchPhrase>
            <field>1245</field>
        </row>
        <row>
            <SearchPhrase>омск</SearchPhrase>
            <field>1112</field>
        </row>
        <row>
            <SearchPhrase>фото собак разных пород</SearchPhrase>
            <field>1091</field>
        </row>
        <row>
            <SearchPhrase>дизайн штор</SearchPhrase>
            <field>1064</field>
        </row>
        <row>
            <SearchPhrase>баку</SearchPhrase>
            <field>1000</field>
        </row>
    </data>
    <rows>10</rows>
    <rows_before_limit_at_least>141137</rows_before_limit_at_least>
</result>

```

Если имя столбца не имеет некоторый допустимый вид, то в качестве имени элемента используется просто `field`. В остальном, структура XML повторяет структуру в формате JSON.

Как и для формата JSON, невалидные UTF-8 последовательности заменяются на replacement character ♦ и, таким образом, выводимый текст будет состоять из валидных UTF-8 последовательностей.

В строковых значениях, экранируются символы < и & как &lt; и &amp;.

Массивы выводятся как `<array><elem>Hello</elem><elem>World</elem>...</array>`, а кортежи как `<tuple><elem>Hello</elem><elem>World</elem>...</tuple>`.

## CapnProto

Cap'n Proto - формат бинарных сообщений, похож на Protocol Buffers и Thrift, но не похож на JSON или MessagePack.

Сообщения Cap'n Proto строго типизированы и не самоописывающиеся, т.е. нуждаются во внешнем описании схемы. Схема применяется «на лету» и кэшируется между запросами.

```
$ cat capnproto_messages.bin | clickhouse-client --query "INSERT INTO test.hits FORMAT CapnProto SETTINGS format_schema='schema:Message'"
```

Где `schema.capnp` выглядит следующим образом:

```
struct Message {  
    SearchPhrase @0 :Text;  
    c @1 :Uint64;  
}
```

Десериализация эффективна и обычно не повышает нагрузку на систему.

См. также [схема формата](#).

## Protobuf

Protobuf - формат [Protocol Buffers](#).

Формат нуждается во внешнем описании схемы. Схема кэшируется между запросами.

ClickHouse поддерживает как синтаксис proto2, так и proto3; все типы полей (repeated/optional/required) поддерживаются.

Пример использования формата:

```
SELECT * FROM test.table FORMAT Protobuf SETTINGS format_schema = 'schemafile:MessageType'
```

или

```
$ cat protobuf_messages.bin | clickhouse-client --query "INSERT INTO test.table FORMAT Protobuf SETTINGS format_schema='schemafile:MessageType'"
```

Где файл `schemafile.proto` может выглядеть так:

```
syntax = "proto3";  
  
message MessageType {  
    string name = 1;  
    string surname = 2;  
    uint32 birthDate = 3;  
    repeated string phoneNumbers = 4;  
};
```

Соответствие между столбцами таблицы и полями сообщения Protocol Buffers устанавливается по имени,

при этом игнорируется регистр букв и символы `_` (подчеркивание) и `.` (точка) считаются одинаковыми.

Если типы столбцов не соответствуют точно типам полей сообщения Protocol Buffers, производится необходимая конвертация.

Вложенные сообщения поддерживаются, например, для поля `z` в таком сообщении

```
message MessageType {  
    message XType {  
        message YType {  
            int32 z;  
        };  
        repeated YType y;  
    };  
    XType x;  
};
```

ClickHouse попытается найти столбец с именем `x.y.z` (или `x_y_z`, или `X.y_Z` и т.п.).

Вложенные сообщения удобно использовать в качестве соответствия для [вложенной структуры данных](#).

Значения по умолчанию, определённые в схеме proto2, например,

```
syntax = "proto2";  
  
message MessageType {  
    optional int32 result_per_page = 3 [default = 10];  
}
```

не применяются; вместо них используются определённые в таблице [значения по умолчанию](#).

ClickHouse пишет и читает сообщения Protocol Buffers в формате `length-delimited`. Это означает, что перед каждым сообщением пишется его длина в формате `varint`. См. также [как читать и записывать сообщения Protocol Buffers в формате length-delimited в различных языках программирования](#).

## ProtobufSingle

То же, что [Protobuf](#), но без разделителей. Позволяет записать / прочитать не более одного сообщения за раз.

## Avro

[Apache Avro](#) — это ориентированный на строки фреймворк для сериализации данных. Разработан в рамках проекта Apache Hadoop.

В ClickHouse формат Avro поддерживает чтение и запись [файлов данных Avro](#).

### Логические типы Avro

## AvroConfluent

Для формата AvroConfluent ClickHouse поддерживает декодирование сообщений Avro с одним объектом. Такие сообщения используются с [Kafka] (<http://kafka.apache.org/>) и реестром схем Confluent.

Каждое сообщение Avro содержит идентификатор схемы, который может быть разрешен для фактической схемы с помощью реестра схем.

Схемы кэшируются после разрешения.

URL-адрес реестра схем настраивается с помощью `format_avro_schema_registry_url`.

## Соответствие типов данных

Такое же, как в [Avro](#).

## Использование

Чтобы быстро проверить разрешение схемы, используйте `kafkacat` с языком запросов `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
```

Чтобы использовать AvroConfluent с `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;
```

## Внимание

`format_avro_schema_registry_url` необходимо настроить в `users.xml`, чтобы сохранить значение после перезапуска. Также можно использовать настройку `format_avro_schema_registry_url` табличного движка `Kafka`.

## Parquet

`Apache Parquet` — формат поколоночного хранения данных, который распространён в экосистеме Hadoop. Для формата `Parquet` ClickHouse поддерживает операции чтения и записи.

### Соответствие типов данных

Таблица ниже содержит поддерживаемые типы данных и их соответствие `типам данных ClickHouse` для запросов `INSERT` и `SELECT`.

Тип данных Parquet (INSERT)	Тип данных ClickHouse	Тип данных Parquet (SELECT)
UINT8, BOOL	UInt8	UInt8
INT8	Int8	INT8
UINT16	UInt16	UInt16
INT16	Int16	INT16
UINT32	UInt32	UInt32

Тип данных Parquet ( <code>INSERT</code> )	Тип данных ClickHouse	Тип данных Parquet ( <code>SELECT</code> )
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

Массивы могут быть вложенными и иметь в качестве аргумента значение типа `Nullable`. Типы `Tuple` и `Map` также могут быть вложенными.

ClickHouse поддерживает настраиваемую точность для формата `Decimal`. При выполнении запроса `INSERT` ClickHouse обрабатывает тип данных Parquet `DECIMAL` как `Decimal128`.

Неподдерживаемые типы данных Parquet: `TIME32`, `FIXED_SIZE_BINARY`, `JSON`, `UUID`, `ENUM`.

Типы данных столбцов в ClickHouse могут отличаться от типов данных соответствующих полей файла в формате Parquet. При вставке данных ClickHouse интерпретирует типы данных в соответствии с таблицей выше, а затем приводит данные к тому типу, который установлен для столбца таблицы.

## Вставка и выборка данных

Чтобы вставить в ClickHouse данные из файла в формате Parquet, выполните команду следующего вида:

```
$ cat {filename} | clickhouse-client --query="INSERT INTO {some_table} FORMAT Parquet"
```

Чтобы вставить данные в колонки типа `Nested` в виде массива структур, нужно включить настройку `input_format_parquet_import_nested`.

Чтобы получить данные из таблицы ClickHouse и сохранить их в файл формата Parquet, используйте команду следующего вида:

```
$ clickhouse-client --query="SELECT * FROM {some_table} FORMAT Parquet" > {some_file.pq}
```

Для обмена данными с экосистемой Hadoop можно использовать движки таблиц [HDFS](#).

## Arrow

[Apache Arrow](#) поставляется с двумя встроенными поколоночными форматами хранения. ClickHouse поддерживает операции чтения и записи для этих форматов.

Arrow — это Apache Arrow's "file mode" формат. Он предназначен для произвольного доступа в памяти.

### Соответствие типов данных

Таблица ниже содержит поддерживаемые типы данных и их соответствие [типам данных ClickHouse](#) для запросов `INSERT` и `SELECT`.

Тип данных Arrow ( <code>INSERT</code> )	Тип данных ClickHouse	Тип данных Arrow ( <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	FixedSize	BINARY
DECIMAL	Decimal	DECIMAL
DECIMAL256	Decimal256	DECIMAL256
LIST	Array	LIST

Тип данных Arrow ( <code>INSERT</code> )	Тип данных ClickHouse	Тип данных Arrow ( <code>SELECT</code> )
STRUCT	Tuple	STRUCT
MAP	Map	MAP

Массивы могут быть вложенными и иметь в качестве аргумента значение типа `Nullable`. Типы `Tuple` и `Map` также могут быть вложенными.

Тип `DICTIONARY` поддерживается для запросов `INSERT`. Для запросов `SELECT` есть настройка `output_format_arrow_low_cardinality_as_dictionary`, которая позволяет выводить тип `LowCardinality` как `DICTIONARY`.

ClickHouse поддерживает настраиваемую точность для формата `Decimal`. При выполнении запроса `INSERT` ClickHouse обрабатывает тип данных Arrow `DECIMAL` как `Decimal128`.

Неподдерживаемые типы данных Arrow: `TIME32`, `FIXED_SIZE_BINARY`, `JSON`, `UUID`, `ENUM`.

Типы данных столбцов в ClickHouse могут отличаться от типов данных соответствующих полей файла в формате Arrow. При вставке данных ClickHouse интерпретирует типы данных в соответствии с таблицей выше, а затем [приводит](#) данные к тому типу, который установлен для столбца таблицы.

## Вставка данных

Чтобы вставить в ClickHouse данные из файла в формате Arrow, используйте команду следующего вида:

```
$ cat filename.arrow | clickhouse-client --query="INSERT INTO some_table FORMAT Arrow"
```

Чтобы вставить данные в колонки типа `Nested` в виде массива структур, нужно включить настройку `input_format_arrow_import_nested`.

## Вывод данных

Чтобы получить данные из таблицы ClickHouse и сохранить их в файл формата Arrow, используйте команду следующего вида:

```
$ clickhouse-client --query="SELECT * FROM {some_table} FORMAT Arrow" > {filename.arrow}
```

## ArrowStream

`ArrowStream` — это Apache Arrow's "stream mode" формат. Он предназначен для обработки потоков в памяти.

## ORC

[Apache ORC](#) — это столбцовый формат данных, распространенный в экосистеме [Hadoop](#).

## Соответствие типов данных

Таблица ниже содержит поддерживаемые типы данных и их соответствие [типам данных ClickHouse](#) для запросов `INSERT` и `SELECT`.

Тип данных ORC ( <code>INSERT</code> )	Тип данных ClickHouse	Тип данных ORC ( <code>SELECT</code> )
<code>UINT8, BOOL</code>	<code>UInt8</code>	<code>UINT8</code>
<code>INT8</code>	<code>Int8</code>	<code>INT8</code>
<code>UINT16</code>	<code>UInt16</code>	<code>UINT16</code>
<code>INT16</code>	<code>Int16</code>	<code>INT16</code>
<code>UINT32</code>	<code>UInt32</code>	<code>UINT32</code>
<code>INT32</code>	<code>Int32</code>	<code>INT32</code>
<code>UINT64</code>	<code>UInt64</code>	<code>UINT64</code>
<code>INT64</code>	<code>Int64</code>	<code>INT64</code>
<code>FLOAT, HALF_FLOAT</code>	<code>Float32</code>	<code>FLOAT</code>
<code>DOUBLE</code>	<code>Float64</code>	<code>DOUBLE</code>
<code>DATE32</code>	<code>Date</code>	<code>DATE32</code>
<code>DATE64, TIMESTAMP</code>	<code>DateTime</code>	<code>TIMESTAMP</code>
<code>STRING, BINARY</code>	<code>String</code>	<code>BINARY</code>
<code>DECIMAL</code>	<code>Decimal</code>	<code>DECIMAL</code>
<code>LIST</code>	<code>Array</code>	<code>LIST</code>
<code>STRUCT</code>	<code>Tuple</code>	<code>STRUCT</code>
<code>MAP</code>	<code>Map</code>	<code>MAP</code>

Массивы могут быть вложенными и иметь в качестве аргумента значение типа `Nullable`. Типы `Tuple` и `Map` также могут быть вложенными.

ClickHouse поддерживает настраиваемую точность для формата `Decimal`. При выполнении запроса `INSERT` ClickHouse обрабатывает тип данных ORC `DECIMAL` как `Decimal128`.

Неподдерживаемые типы данных ORC: `TIME32, FIXED_SIZE_BINARY, JSON, UUID, ENUM`.

Типы данных столбцов в таблицах ClickHouse могут отличаться от типов данных для соответствующих полей ORC. При вставке данных ClickHouse интерпретирует типы данных ORC согласно таблице соответствия, а затем [приводит](#) данные к типу, установленному для столбца таблицы ClickHouse.

## Вставка данных

Чтобы вставить в ClickHouse данные из файла в формате ORC, используйте команду следующего вида:

```
$ cat filename.orc | clickhouse-client --query="INSERT INTO some_table FORMAT ORC"
```

Чтобы вставить данные в колонки типа **Nested** в виде массива структур, нужно включить настройку **input\_format\_orc\_import\_nested**.

## Вывод данных

Чтобы получить данные из таблицы ClickHouse и сохранить их в файл формата ORC, используйте команду следующего вида:

```
$ clickhouse-client --query="SELECT * FROM {some_table} FORMAT ORC" > {filename.orc}
```

Для обмена данных с экосистемой Hadoop вы можете использовать [движок таблиц HDFS](#).

## LineAsString

В этом формате каждая строка импортируемых данных интерпретируется как одно строковое значение. Парситься может только таблица с единственным полем типа **String**. Остальные столбцы должны быть заданы как **DEFAULT** или **MATERIALIZED**, либо отсутствовать.

### Пример

Запрос:

```
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;
```

Результат:

```
field
"I love apple", "I love banana", "I love orange"; |
```

## Regexp

Каждая строка импортируемых данных разбирается в соответствии с регулярным выражением.

При работе с форматом **Regexp** можно использовать следующие параметры:

- **format\_regexp** — **String**. Стока с регулярным выражением в формате **re2**.
- **format\_regexp\_escaping\_rule** — **String**. Правило сериализации. Поддерживаются следующие правила:
  - CSV (как в **CSV**)
  - JSON (как в **JSONEachRow**)
  - Escaped (как в **TSV**)
  - Quoted (как в **Values**)
  - Raw (данные импортируются как есть, без сериализации)
- **format\_regexp\_skip\_unmatched** — **UInt8**. Признак, будет ли генерироваться исключение в случае, если импортируемые данные не соответствуют регулярному выражению **format\_regexp**. Может принимать значение 0 или 1.

## Использование

Регулярное выражение (шаблон) из параметра `format_regex` применяется к каждой строке импортируемых данных. Количество частей в шаблоне (подшаблонов) должно соответствовать количеству колонок в импортируемых данных.

Строки импортируемых данных должны разделяться символом новой строки "`\n`" или символами "`\r\n`" (перенос строки в формате DOS).

Данные, выделенные по подшаблонам, интерпретируются в соответствии с типом, указанным в параметре `format_regex_escaping_rule`.

Если строка импортируемых данных не соответствует регулярному выражению и параметр `format_regex_skip_unmatched` равен 1, строка просто игнорируется. Если же параметр `format_regex_skip_unmatched` равен 0, генерируется исключение.

## Пример

Рассмотрим файл `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
```

и таблицу:

```
CREATE TABLE imp_regex_table (id UInt32, array Array(UInt32), string String, date Date) ENGINE = Memory;
```

Команда импорта:

```
$ 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;"
```

Запрос:

```
SELECT * FROM imp_regex_table;
```

Результат:

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

## Схема формата

Имя файла со схемой записывается в настройке `format_schema`. При использовании форматов `Cap'n Proto` и `Protobuf` требуется указать схему.

Схема представляет собой имя файла и имя типа в этом файле, разделенные двоеточием, например `schemafile.proto:MessageType`.

Если файл имеет стандартное расширение для данного формата (например `.proto` для `Protobuf`), то можно его не указывать и записывать схему так `schemafile:MessageType`.

Если для ввода/вывода данных используется [клиент в интерактивном режиме](#), то при записи схемы можно использовать абсолютный путь или записывать путь относительно текущей директории на клиенте. Если клиент используется в [batch режиме](#), то в записи схемы допускается только относительный путь, из соображений безопасности.

Если для ввода/вывода данных используется [HTTP-интерфейс](#), то файл со схемой должен располагаться на сервере в каталоге, указанном в параметре [format\\_schema\\_path](#) конфигурации сервера.

## Игнорирование ошибок

Некоторые форматы, такие как CSV, TabSeparated, TSKV, JSONEachRow, Template, CustomSeparated и Protobuf, могут игнорировать строки, которые не соответствуют правилам и разбор которых может вызвать ошибку. При этом обработка импортируемых данных продолжается со следующей строки. См. настройки [input\\_format\\_allow\\_errors\\_num](#) и [input\\_format\\_allow\\_errors\\_ratio](#).

Ограничения:

- В формате `JSONEachRow` в случае ошибки игнорируются все данные до конца текущей строки (или до конца файла). Поэтому строки должны быть разделены символом `\n`, чтобы ошибки обрабатывались корректно.
- Форматы `Template` и `CustomSeparated` используют разделитель после последней колонки и разделитель между строками. Поэтому игнорирование ошибок работает только если хотя бы одна из строк не пустая.

## RawBLOB

В этом формате все входные данныечитываются в одно значение. Парсить можно только таблицу с одним полем типа [String](#) или подобным ему.

Результат выводится в бинарном виде без разделителей и экранирования. При выводе более одного значения формат неоднозначен и будет невозможно прочитать данные снова.

Ниже приведено сравнение форматов `RawBLOB` и `TabSeparatedRaw`.

`RawBLOB`:

- данные выводятся в бинарном виде, без экранирования;
- нет разделителей между значениями;
- нет перевода строки в конце каждого значения.

`TabSeparatedRaw`:

- данные выводятся без экранирования;
- строка содержит значения, разделённые табуляцией;
- после последнего значения в строке есть перевод строки.

Далее рассмотрено сравнение форматов `RawBLOB` и `RowBinary`.

`RawBLOB`:

- строки выводятся без их длины в начале.

`RowBinary`:

- строки представлены как длина в формате varint (unsigned [LEB128](#)), а затем байты строки.

При передаче на вход `RawBLOB` пустых данных, ClickHouse бросает исключение:

```
Code: 108. DB::Exception: No data to insert
```

## Пример

```
$ 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
```

Результат:

```
f9725a22f9191e064120d718e26862a9 -
```

## MsgPack

ClickHouse поддерживает запись и чтение из файлов в формате [MessagePack](#).

### Соответствие типов данных

Тип данных MsgPack	Тип данных ClickHouse
uint N, positive fixint	UIntN
int N	IntN
fixstr, str 8, str 16, str 32	String, FixedString
float 32	Float32
float 64	Float64
uint 16	Date
uint 32	DateTime
uint 64	DateTime64
fixarray, array 16, array 32	Array
nil	Nothing

Пример:

Запись в файл ".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-драйвер

- [Официальный драйвер](#)
- Драйверы от сторонних организаций:
  - [ClickHouse-Native-JDBC](#)
  - [clickhouse4j](#)

## ODBC-драйвер

- Официальный драйвер.

## C++ клиентская библиотека

См. README в репозитории [clickhouse-cpp](#).

## Сторонние интерфейсы

Раздел содержит список сторонних интерфейсов для ClickHouse. Это может быть визуальный интерфейс, интерфейс командной строки, либо API:

- [Client libraries](#)
- [Integrations](#)
- [GUI](#)
- [Proxies](#)

### Примечание

С ClickHouse работают также универсальные инструменты, поддерживающие общий API, такие как **ODBC** или **JDBC**.

## Клиентские библиотеки от сторонних разработчиков

### Disclaimer

Яндекс не поддерживает перечисленные ниже библиотеки и не проводит тщательного тестирования для проверки их качества.

- 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](#)
  - [glushkovds/phpclickhouse-laravel](#)
  - [kolya7k ClickHouse PHP extension](#)
- Go
  - [clickhouse](#)
  - [go-clickhouse](#)
  - [mailrugo-clickhouse](#)
  - [golang-clickhouse](#)
- 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](#)
- Scala
  - [clickhouse-scala-client](#)
- Kotlin
  - [AORM](#)

- C#
  - Octonica.ClickHouseClient
  - ClickHouse.Ado
  - ClickHouse.Client
  - ClickHouse.Net
- Elixir
  - clickhousex
  - pillar
- Nim
  - nim-clickhouse

## Библиотеки для интеграции от сторонних разработчиков

### Disclaimer

Яндекс не занимается поддержкой перечисленных ниже инструментов и библиотек и не проводит тщательного тестирования для проверки их качества.

## Инфраструктурные продукты

- Реляционные системы управления базами данных
  - MySQL
    - mysql2ch
    - ProxySQL
    - clickhouse-mysql-data-reader
    - horgh-replicator
  - PostgreSQL
    - clickhousedb\_fdw
    - infi.clickhouse\_fdw (использует infi.clickhouse\_orm)
    - pg2ch
    - clickhouse\_fdw
  - MSSQL
    - ClickHouseMigrator
- Очереди сообщений
  - Kafka
    - clickhouse\_sinker (использует Go client)
    - stream-loader-clickhouse

- Потоковая обработка
  - [Flink](#)
    - [flink-clickhouse-sink](#)
- Хранилища объектов
  - [S3](#)
    - [clickhouse-backup](#)
- Оркестрация контейнеров
  - [Kubernetes](#)
    - [clickhouse-operator](#)
- Системы управления конфигурацией
  - [puppet](#)
    - [innogames/clickhouse](#)
    - [mfedotov/clickhouse](#)
- Мониторинг
  - [Graphite](#)
    - [graphhouse](#)
    - [carbon-clickhouse](#)
    - [graphite-clickhouse](#)
  - [graphite-ch-optimizer](#) - оптимизирует партиции таблиц [\\*GraphiteMergeTree](#) согласно правилам в конфигурации rollup
  - [Grafana](#)
    - [clickhouse-grafana](#)
  - [Prometheus](#)
    - [clickhouse\\_exporter](#)
    - [PromHouse](#)
    - [clickhouse\\_exporter](#) (использует [Go client](#))
  - [Nagios](#)
    - [check\\_clickhouse](#)
    - [check\\_clickhouse.py](#)
  - [Zabbix](#)
    - [clickhouse-zabbix-template](#)
  - [Semantext](#)
    - [clickhouse интеграция](#)
- Логирование
  - [rsyslog](#)
    - [omclickhouse](#)
  - [fluentd](#)
    - [loghouse](#) (для [Kubernetes](#))
  - [Semantext](#)
    - [logagent output-plugin-clickhouse](#)

- Гео
  - MaxMind
    - clickhouse-maxmind-geoip
- AutoML
  - MindsDB
    - MindsDB - Слой предиктивной аналитики и искусственного интеллекта для СУБД ClickHouse.

## Экосистемы вокруг языков программирования

- Python
  - SQLAlchemy
    - sqlalchemy-clickhouse (использует infi.clickhouse\_orm)
  - pandas
    - pandahouse
- PHP
  - Doctrine
    - dbal-clickhouse
- R
  - dplyr
  - RClickhouse (использует clickhouse-cpp)
- Java
  - Hadoop
  - clickhouse-hdfs-loader (использует 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

# Визуальные интерфейсы от сторонних разработчиков

## С открытым исходным кодом

### Tabix

Веб-интерфейс для ClickHouse в проекте [Tabix](#).

Основные возможности:

- Работает с ClickHouse напрямую из браузера, без необходимости установки дополнительного ПО;
- Редактор запросов с подсветкой синтаксиса;
- Автодополнение команд;
- Инструменты графического анализа выполнения запросов;
- Цветовые схемы на выбор.

[Документация Tabix](#).

### HouseOps

[HouseOps](#) — UI/IDE для OSX, Linux и Windows.

Основные возможности:

- Построение запросов с подсветкой синтаксиса;
- Просмотр ответа в табличном или JSON представлении;
- Экспортирование результатов запроса в формате CSV или JSON;
- Список процессов с описанием;
- Режим записи;
- Возможность остановки (KILL) запроса;
- Граф базы данных. Показывает все таблицы и их столбцы с дополнительной информацией;
- Быстрый просмотр размера столбца;
- Конфигурирование сервера.

Планируется разработка следующих возможностей:

- Управление базами;
- Управление пользователями;
- Анализ данных в режиме реального времени;
- Мониторинг кластера;
- Управление кластером;
- Мониторинг реплицированных и Kafka таблиц.

### LightHouse

[LightHouse](#) — это легковесный веб-интерфейс для ClickHouse.

Основные возможности:

- Список таблиц с фильтрацией и метаданными;
- Предварительный просмотр таблицы с фильтрацией и сортировкой;
- Выполнение запросов только для чтения.

## Redash

[Redash](#) — платформа для отображения данных.

Поддерживает множество источников данных, включая ClickHouse. Redash может объединять результаты запросов из разных источников в финальный набор данных.

Основные возможности:

- Мощный редактор запросов.
- Проводник по базе данных.
- Инструменты визуализации, позволяющие представить данные в различных формах.

## DBeaver

[DBeaver](#) - универсальный desktop клиент баз данных с поддержкой ClickHouse.

Основные возможности:

- Построение запросов с подсветкой синтаксиса.
- Просмотр таблиц.
- Автодополнение команд.
- Полнотекстовый поиск.

По умолчанию DBeaver не использует сессии при подключении (в отличие от CLI, например). Если вам нужна поддержка сессий (например, для установки настроек на сессию), измените настройки подключения драйвера и укажите для настройки `session_id` любое произвольное значение (драйвер использует подключение по http). После этого вы можете использовать любую настройку (setting) в окне запроса.

## clickhouse-cli

[clickhouse-cli](#) - это альтернативный клиент командной строки для ClickHouse, написанный на Python 3.

Основные возможности:

- Автодополнение;
- Подсветка синтаксиса для запросов и вывода данных;
- Поддержка постраничного просмотра для результирующих данных;
- Дополнительные PostgreSQL-подобные команды.

## clickhouse-flamegraph

[clickhouse-flamegraph](#) — специализированный инструмент для визуализации `system.trace_log` в виде [flamegraph](#).

## clickhouse-plantuml

[clickhouse-plantuml](#) — скрипт, генерирующий [PlantUML](#) диаграммы схем таблиц.

## xeus-clickhouse

[xeus-clickhouse](#) — это ядро Jupyter для ClickHouse, которое поддерживает запрос ClickHouse-данных с использованием SQL в Jupyter.

## MindsDB Studio

[MindsDB](#) — это продукт с открытым исходным кодом, реализующий слой искусственного интеллекта (Artificial Intelligence, AI) для различных СУБД, в том числе для ClickHouse. MindsDB облегчает процессы создания, обучения и развертывания современных моделей машинного обучения. Графический пользовательский интерфейс MindsDB Studio позволяет обучать новые модели на основе данных в БД, интерпретировать сделанные моделями прогнозы, выявлять потенциальные ошибки в данных, визуализировать и оценивать достоверность моделей с помощью функции Explainable AI, так чтобы вы могли быстрее адаптировать и настраивать ваши модели машинного обучения.

## Коммерческие

### DataGrip

[DataGrip](#) — это IDE для баз данных от JetBrains с выделенной поддержкой ClickHouse. Он также встроен в другие инструменты на основе IntelliJ: PyCharm, IntelliJ IDEA, GoLand, PhpStorm и другие.

Основные возможности:

- Очень быстрое дополнение кода.
- Подсветка синтаксиса для SQL диалекта ClickHouse.
- Поддержка функций, специфичных для ClickHouse, например вложенных столбцов, движков таблиц.
- Редактор данных.
- Рефакторинги.
- Поиск и навигация.

## Yandex DataLens

[Yandex DataLens](#) — сервис визуализации и анализа данных.

Основные возможности:

- Широкий выбор инструментов визуализации, от простых столбчатых диаграмм до сложных дашбордов.
- Возможность опубликовать дашборды на широкую аудиторию.
- Поддержка множества источников данных, включая ClickHouse.
- Хранение материализованных данных в кластере ClickHouse DataLens.

Для небольших проектов DataLens [доступен бесплатно](#), в том числе и для коммерческого использования.

- [Документация DataLens](#).
- [Пособие по визуализации данных из ClickHouse](#).

## Holistics Software

[Holistics](#) — full-stack платформа для обработки данных и бизнес-аналитики.

Основные возможности:

- Автоматизированные отчёты на почту, Slack, и Google Sheet.
- Редактор SQL с визуализацией, контролем версий, автодополнением, повторным использованием частей запроса и динамическими фильтрами.
- Встроенные инструменты анализа отчётов и всплывающие (iframe) дашборды.
- Подготовка данных и возможности ETL.
- Моделирование данных с помощью SQL для их реляционного отображения.

## Looker

[Looker](#) — платформа для обработки данных и бизнес-аналитики. Поддерживает более 50 диалектов баз данных, включая ClickHouse. Looker можно установить самостоятельно или воспользоваться готовой платформой SaaS.

Просмотр данных, построение отображений и дашбордов, планирование отчётов и обмен данными с коллегами доступны с помощью браузера. Также, Looker предоставляет ряд инструментов, позволяющих встраивать сервис в другие приложения и API для обмена данными.

Основные возможности:

- Язык LookML, поддерживающий [моделирование данных](#).
- Интеграция с различными системами с помощью [Data Actions](#).
- Инструменты для встраивания сервиса в приложения.
- API.

[Как сконфигурировать ClickHouse в Looker](#).

## SeekTable

[SeekTable](#) — это аналитический инструмент для самостоятельного анализа и обработки данных бизнес-аналитики. Он доступен как в виде облачного сервиса, так и в виде локальной версии. Отчеты из SeekTable могут быть встроены в любое веб-приложение.

Основные возможности:

- Удобный конструктор отчетов.
- Гибкая настройка отчетов SQL и создание запросов для специфичных отчетов.
- Интегрируется с ClickHouse, используя собственную точку приема запроса TCP/IP или интерфейс HTTP(S) (два разных драйвера).

- Поддерживает всю мощь диалекта ClickHouse SQL для построения запросов по различным измерениям и показателям.
- **WEB-API** для автоматизированной генерации отчетов.
- Процесс разработки отчетов поддерживает **резервное копирование/восстановление данных**; конфигурация моделей данных (кубов) / отчетов представляет собой удобочитаемый XML-файл, который может храниться в системе контроля версий.

SeekTable **бесплатен** для личного/индивидуального использования.

[Как сконфигурировать подключение ClickHouse в SeekTable.](#)

## Chadmin

**Chadmin** — простой графический интерфейс для визуализации запущенных запросов на вашем кластере ClickHouse. Он отображает информацию о запросах и дает возможность их завершать.

## Прокси-серверы от сторонних разработчиков chproxy

**chproxy** - это http-прокси и балансировщик нагрузки для базы данных ClickHouse.

Основные возможности:

- Индивидуальная маршрутизация и кэширование ответов;
- Гибкие ограничения;
- Автоматическое продление SSL сертификатов.

Реализован на Go.

## KittenHouse

**KittenHouse** предназначен для использования в качестве локального прокси-сервера между ClickHouse и вашим сервером приложений в случае, если буферизовать данные INSERT на стороне приложения не представляется возможным или не удобно.

Основные возможности:

- Буферизация данных в памяти и на диске;
- Маршрутизация по таблицам;
- Балансировка нагрузки и проверка работоспособности.

Реализован на Go.

## ClickHouse-Bulk

**ClickHouse-Bulk** - простой сборщик вставок ClickHouse.

Особенности:

- Группировка запросов и отправка по порогу или интервалу;
- Несколько удаленных серверов;
- Базовая аутентификация.

## Движки таблиц

Движок таблицы (тип таблицы) определяет:

- Как и где хранятся данные, куда их писать и откуда читать.
- Какие запросы поддерживаются и каким образом.
- Конкурентный доступ к данным.
- Использование индексов, если есть.
- Возможно ли многопоточное выполнение запроса.
- Параметры репликации данных.

## Семейства движков

### MergeTree

Наиболее универсальные и функциональные движки таблиц для задач с высокой загрузкой. Общим свойством этих движков является быстрая вставка данных с последующей фоновой обработкой данных. Движки \*MergeTree поддерживают репликацию данных (в Replicated\* версиях движков), партиционирование, и другие возможности не поддержанные для других движков.

Движки семейства:

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

### Log

Простые [движки](#) с минимальной функциональностью. Они наиболее эффективны, когда вам нужно быстро записать много небольших таблиц (до примерно 1 миллиона строк) и прочитать их позже целиком.

Движки семейства:

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

## Движки для интеграции

Движки для связи с другими системами хранения и обработки данных.

Движки семейства:

- Kafka
- MySQL
- ODBC
- JDBC
- S3

## Специальные движки

- ODBC
- JDBC
- MySQL
- MongoDB
- HDFS
- Kafka
- EmbeddedRocksDB
- RabbitMQ
- PostgreSQL

## Специальные движки

Движки семейства:

- Distributed
- MaterializedView
- Dictionary
- Merge
- File
- Null
- Set
- Join
- URL
- View
- Memory
- Buffer

## Виртуальные столбцы

Виртуальный столбец — это неотъемлемый атрибут движка таблиц, определенный в исходном коде движка.

Виртуальные столбцы не надо указывать в запросе `CREATE TABLE` и они не отображаются в результатах запросов `SHOW CREATE TABLE` и `DESCRIBE TABLE`. Также виртуальные столбцы доступны только для чтения, поэтому вы не можете вставлять в них данные.

Чтобы получить данные из виртуального столбца, необходимо указать его название в запросе `SELECT`. `SELECT *` не отображает данные из виртуальных столбцов.

При создании таблицы со столбцом, имя которого совпадает с именем одного из виртуальных столбцов таблицы, виртуальный столбец становится недоступным. Не делайте так. Чтобы помочь избежать конфликтов, имена виртуальных столбцов обычно предваряются подчёркиванием.

## MergeTree

Движок MergeTree, а также другие движки этого семейства (`*MergeTree`) — это наиболее функциональные движки таблиц ClickHouse.

Основная идея, заложенная в основу движков семейства MergeTree следующая. Когда у вас есть огромное количество данных, которые должны быть вставлены в таблицу, вы должны быстро записать их по частям, а затем объединить части по некоторым правилам в фоновом режиме. Этот метод намного эффективнее, чем постоянная перезапись данных в хранилище при вставке.

Основные возможности:

- **Хранит данные, отсортированные по первичному ключу.** Это позволяет создавать разреженный индекс небольшого объёма, который позволяет быстрее находить данные.
- **Позволяет оперировать партициями, если задан `ключ партиционирования`.** ClickHouse поддерживает отдельные операции с партициями, которые работают эффективнее, чем общие операции с этим же результатом над этими же данными. Также, ClickHouse автоматически отсекает данные по партициям там, где ключ партиционирования указан в запросе. Это также увеличивает эффективность выполнения запросов.
- **Поддерживает репликацию данных.** Для этого используется семейство таблиц `ReplicatedMergeTree`. Подробнее читайте в разделе [Репликация данных](#).
- **Поддерживает сэмплирование данных.** При необходимости можно задать способ сэмплирования данных в таблице.

### Info

Движок **Merge** не относится к семейству `*MergeTree`.

## Создание таблицы

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
(
    name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1] [TTL expr1],
    name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2] [TTL expr2],
    ...
    INDEX index_name1 expr1 TYPE type1(...) GRANULARITY value1,
    INDEX index_name2 expr2 TYPE type2(...) GRANULARITY value2
) ENGINE = MergeTree()
ORDER BY expr
[PARTITION BY expr]
[PRIMARY KEY expr]
[SAMPLE BY expr]
[TTL expr
    [DELETE|TO DISK 'xxx'|TO VOLUME 'xxx' [, ...]]
    [WHERE conditions]
    [GROUP BY key_expr [SET v1 = aggr_func(v1) [, v2 = aggr_func(v2) ...]]]]
[SETTINGS name=value, ...]
```

Описание параметров смотрите в [описании запроса CREATE](#).

## Секции запроса

- **ENGINE** — имя и параметры движка. `ENGINE = MergeTree()`. `MergeTree` не имеет параметров.
- **ORDER BY** — ключ сортировки.

Кортеж столбцов или произвольных выражений. Пример: `ORDER BY (CounterID, EventDate)`.

ClickHouse использует ключ сортировки в качестве первичного ключа, если первичный ключ не задан в секции `PRIMARY KEY`.

Чтобы отключить сортировку, используйте синтаксис `ORDER BY tuple()`. Смотрите [выбор первичного ключа](#).

- **PARTITION BY** — [ключpartitionирования](#). Необязательный параметр.

Для partitionирования по месяцам используйте выражение `toYYYYMM(date_column)`, где `date_column` — столбец с датой типа `Date`. В этом случае имена партиций имеют формат "YYYYMM".

- **PRIMARY KEY** — первичный ключ, если он [отличается от ключа сортировки](#). Необязательный параметр.

По умолчанию первичный ключ совпадает с ключом сортировки (который задаётся секцией `ORDER BY`.) Поэтому в большинстве случаев секцию `PRIMARY KEY` отдельно указывать не нужно.

- **SAMPLE BY** — выражение для сэмплирования. Необязательный параметр.

Если используется выражение для сэмплирования, то первичный ключ должен содержать его. Результат выражения для сэмплирования должен быть беззнаковым целым числом. Пример: `SAMPLE BY intHash32(UserID) ORDER BY (CounterID, EventDate, intHash32(UserID))`

- **TTL** — список правил, определяющих длительности хранения строк, а также задающих правила перемещения частей на определённые тома или диски. Необязательный параметр.

Выражение должно возвращать столбец `Date` или `DateTime`. Пример: `TTL date + INTERVAL 1 DAY`.

Тип правила `DELETE|TO DISK 'xxx'|TO VOLUME 'xxx'|GROUP BY` указывает действие, которое будет выполнено с частью: удаление строк (прореживание), перемещение (при выполнении условия для всех строк части) на определённый диск (`TO DISK 'xxx'`) или том (`TO VOLUME 'xxx'`), или агрегирование данных в устаревших строках. Поведение по умолчанию соответствует удалению строк (`DELETE`). В списке правил может быть указано только одно выражение с поведением `DELETE`.

Дополнительные сведениясмотрите в разделе [TTL для столбцов и таблиц](#)

- **SETTINGS** — дополнительные параметры, регулирующие поведение `MergeTree` (необязательные):
  - `index_granularity` — максимальное количество строк данных между засечками индекса. По умолчанию — 8192. Смотрите [Хранение данных](#).
  - `index_granularity_bytes` — максимальный размер гранул данных в байтах. По умолчанию — 10Mb. Чтобы ограничить размер гранул только количеством строк, установите значение 0 (не рекомендовано). Смотрите [Хранение данных](#).
  - `min_index_granularity_bytes` — минимально допустимый размер гранул данных в байтах. Значение по умолчанию — 1024b. Для обеспечения защиты от случайного создания таблиц с очень низким значением `index_granularity_bytes`. Смотрите [Хранение данных](#).
  - `enable_mixed_granularity_parts` — включает или выключает переход к ограничению размера гранул с помощью настройки `index_granularity_bytes`. Настройка `index_granularity_bytes` улучшает производительность ClickHouse при выборке данных из таблиц с большими (десятки и сотни мегабайтов) строками. Если у вас есть таблицы с большими строками, можно включить эту настройку, чтобы повысить эффективность запросов `SELECT`.
  - `use_minimalistic_part_header_in_zookeeper` — Способ хранения заголовков кусков данных в ZooKeeper. Если `use_minimalistic_part_header_in_zookeeper = 1`, то ZooKeeper хранит меньше данных. Подробнее читайте в [описании настройки](#) в разделе "Конфигурационные параметры сервера".
  - `min_merge_bytes_to_use_direct_io` — минимальный объём данных при слиянии, необходимый для прямого (небуферизованного) чтения/записи (direct I/O) на диск. При слиянии частей данных ClickHouse вычисляет общий объём хранения всех данных, подлежащих слиянию. Если общий объём хранения всех данных для чтения превышает `min_bytes_to_use_direct_io` байт, тогда ClickHouse использует флаг `O_DIRECT` при чтении данных с диска. Если `min_merge_bytes_to_use_direct_io = 0`, тогда прямой ввод-вывод отключен. Значение по умолчанию: `10 * 1024 * 1024 * 1024` байтов.
  - `merge_with_ttl_timeout` — минимальное время в секундах перед повторным слиянием для удаления данных с истекшим TTL. По умолчанию: `14400` секунд (4 часа).
  - `merge_with_recompression_ttl_timeout` — минимальное время в секундах перед повторным слиянием для повторного сжатия данных с истекшим TTL. По умолчанию: `14400` секунд (4 часа).
  - `try_fetch_recompressed_part_timeout` — время ожидания (в секундах) перед началом слияния с повторным сжатием. В течение этого времени ClickHouse пытается извлечь сжатую часть из реплики, которая назначила это слияние. Значение по умолчанию: `7200` секунд (2 часа).
  - `write_final_mark` — включает или отключает запись последней засечки индекса в конце куска данных, указывающей за последний байт. По умолчанию — 1. Не отключайте её.

- `merge_max_block_size` — максимальное количество строк в блоке для операций слияния. Значение по умолчанию: 8192.
- `storage_policy` — политика хранения данных. Смотрите [Хранение данных таблицы на нескольких блочных устройствах](#).
- `min_bytes_for_wide_part`, `min_rows_for_wide_part` — минимальное количество байт/строк в куске данных для хранения в формате `Wide`. Можно задать одну или обе настройки или не задавать ни одной. Подробнее см. в разделе [Хранение данных](#).
- `max_parts_in_total` — максимальное количество кусков во всехパーティциях.
- `max_compress_block_size` — максимальный размер блоков несжатых данных перед сжатием для записи в таблицу. Вы также можете задать этот параметр в глобальных настройках (смотрите [max\\_compress\\_block\\_size](#)). Настройка, которая задается при создании таблицы, имеет более высокий приоритет, чем глобальная.
- `min_compress_block_size` — минимальный размер блоков несжатых данных, необходимых для сжатия при записи следующей засечки. Вы также можете задать этот параметр в глобальных настройках (смотрите [min\\_compress\\_block\\_size](#)). Настройка, которая задается при создании таблицы, имеет более высокий приоритет, чем глобальная.
- `max_partitions_to_read` — Ограничивает максимальное число партиций для чтения в одном запросе. Также возможно указать настройку [max\\_partitions\\_to\\_read](#) в глобальных настройках.

## Пример задания секций

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

В примере мы устанавливаем партиционирование по месяцам.

Также мы задаем выражение для сэмплирования в виде хэша по идентификатору посетителя. Это позволяет псевдослучайным образом перемешать данные в таблице для каждого `CounterID` и `EventDate`. Если при выборке данных задать секцию `SAMPLE`, то ClickHouse вернёт равномерно-псевдослучайную выборку данных для подмножества посетителей.

`index_granularity` можно было не указывать, поскольку 8192 — это значение по умолчанию.

### ► Устаревший способ создания таблицы

## Хранение данных

Таблица состоит из кусков данных (data parts), отсортированных по первичному ключу.

При вставке в таблицу создаются отдельные куски данных, каждый из которых лексикографически отсортирован по первичному ключу. Например, если первичный ключ — `(CounterID, Date)`, то данные в куске будут лежать в порядке `CounterID`, а для каждого `CounterID` в порядке `Date`.

Данные, относящиеся к разным партициям, разбиваются на разные куски. В фоновом режиме ClickHouse выполняет слияния (merge) кусков данных для более эффективного хранения. Куски, относящиеся к разным партициям не объединяются. Механизм слияния не гарантирует, что все строки с одинаковым первичным ключом окажутся в одном куске.

Куски данных могут храниться в формате `Wide` или `Compact`. В формате `Wide` каждый столбец хранится в отдельном файле, а в формате `Compact` все столбцы хранятся в одном файле. Формат `Compact` может быть полезен для повышения производительности при частом добавлении небольших объемов данных.

Формат хранения определяется настройками движка `min_bytes_for_wide_part` и `min_rows_for_wide_part`. Если число байт или строк в куске данных меньше значения, указанного в соответствующей настройке, тогда этот кусок данных хранится в формате `Compact`. В противном случае кусок данных хранится в формате `Wide`. Если ни одна из настроек не задана, куски данных хранятся в формате `Wide`.

Каждый кусок данных логически делится на гранулы. Гранула — это минимальный неделимый набор данных, который ClickHouse считывает при выборке данных. ClickHouse не разбивает строки и значения и гранула всегда содержит целое число строк. Первая строка гранулы помечается значением первичного ключа для этой строки (засечка). Для каждого куска данных ClickHouse создаёт файл с засечками (индексный файл). Для каждого столбца, независимо от того, входит он в первичный ключ или нет, ClickHouse также сохраняет эти же засечки. Засечки используются для поиска данных напрямую в файлах столбцов.

Размер гранул ограничен настройками движка `index_granularity` и `index_granularity_bytes`. Количество строк в грануле лежит в диапазоне  $[1, \text{index\_granularity}]$ , в зависимости от размера строк. Размер гранулы может превышать `index_granularity_bytes` в том случае, когда размер единственной строки в грануле превышает значение настройки. В этом случае, размер гранулы равен размеру строки.

## Первичные ключи и индексы в запросах

Рассмотрим первичный ключ — (`CounterID`, `Date`). В этом случае сортировку и индекс можно проиллюстрировать следующим образом:

```
Whole data: [-----]
CounterID: [aaaaaaaaaaaaaaaaabbbbcdeeeeeeeeeefggggggghhhhhhhiiiiiiiklllllll]
Date:      [11111112222223331233211112222233211111121222222311112223311122333]
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
```

Если в запросе к данным указать:

- `CounterID IN ('a', 'h')`, то сервер читает данные в диапазонах засечек `[0, 3]` и `[6, 8]`.
- `CounterID IN ('a', 'h') AND Date = 3`, то сервер читает данные в диапазонах засечек `[1, 3]` и `[7, 8]`.
- `Date = 3`, то сервер читает данные в диапазоне засечек `[1, 10]`.

Примеры выше показывают, что использование индекса всегда эффективнее, чем `full scan`.

Разреженный индекс допускает чтение лишних строк. При чтении одного диапазона первичного ключа, может быть прочитано до `index_granularity * 2` лишних строк в каждом блоке данных.

Разреженный индекс почти всегда помещается в оперативную память и позволяет работать с очень большим количеством строк в таблицах.

ClickHouse не требует уникального первичного ключа. Можно вставить много строк с одинаковым первичным ключом.

Ключ в `PRIMARY KEY` и `ORDER BY` может иметь тип `Nullable`. За поддержку этой возможности отвечает настройка `allow_nullable_key`.

При сортировке с использованием выражения `ORDER BY` для значений `NULL` всегда работает принцип `NULLS_LAST`.

## Выбор первичного ключа

Количество столбцов в первичном ключе не ограничено явным образом. В зависимости от структуры данных в первичный ключ можно включать больше или меньше столбцов. Это может:

- Увеличить эффективность индекса.

Пусть первичный ключ —  $(a, b)$ , тогда добавление ещё одного столбца с повысит эффективность, если выполнены условия:

- Есть запросы с условием на столбец  $c$ .
- Часто встречаются достаточно длинные (в несколько раз больше `index_granularity`) диапазоны данных с одинаковыми значениями  $(a, b)$ . Иначе говоря, когда добавление ещё одного столбца позволит пропускать достаточно длинные диапазоны данных.
- Улучшить сжатие данных.

ClickHouse сортирует данные по первичному ключу, поэтому чем выше однородность, тем лучше сжатие.

- Обеспечить дополнительную логику при слиянии кусков данных в движках `CollapsingMergeTree` и `SummingMergeTree`.

В этом случае имеет смысл указать отдельный ключ сортировки, отличающийся от первичного ключа.

Длинный первичный ключ будет негативно влиять на производительность вставки и потребление памяти, однако на производительность ClickHouse при запросах `SELECT` лишние столбцы в первичном ключе не влияют.

Вы можете создать таблицу без первичного ключа, используя синтаксис `ORDER BY tuple()`. В этом случае ClickHouse хранит данные в порядке вставки. Если вы хотите сохранить порядок данных при вставке данных с помощью запросов `INSERT ... SELECT`, установите `max_insert_threads = 1`.

Чтобы выбрать данные в первоначальном порядке, используйте **однопоточные** запросы `SELECT`.

## Первичный ключ, отличный от ключа сортировки

Существует возможность задать первичный ключ (выражение, значения которого будут записаны в индексный файл для каждой засечки), отличный от ключа сортировки (выражение, по которому будут упорядочены строки в кусках данных). Кортеж выражения первичного ключа при этом должен быть префиксом кортежа выражения ключа сортировки.

Данная возможность особенно полезна при использовании движков [SummingMergeTree](#) и [AggregatingMergeTree](#). В типичном сценарии использования этих движков таблица содержит столбцы двух типов: *измерения* (dimensions) и *меры* (measures). Типичные запросы агрегируют значения столбцов-мер с произвольной группировкой и фильтрацией по измерениям. Так как [SummingMergeTree](#) и [AggregatingMergeTree](#) производят фоновую агрегацию строк с одинаковым значением ключа сортировки, приходится добавлять в него все столбцы-измерения. В результате выражение ключа содержит большой список столбцов, который приходится постоянно расширять при добавлении новых измерений.

В этом сценарии имеет смысл оставить в первичном ключе всего несколько столбцов, которые обеспечат эффективную фильтрацию по индексу, а остальные столбцы-измерения добавить в выражение ключа сортировки.

[ALTER ключа сортировки](#) — лёгкая операция, так как при одновременном добавлении нового столбца в таблицу и ключ сортировки не нужно изменять данные кусков (они остаются упорядоченными и по новому выражению ключа).

## Использование индексов и партиций в запросах

Для запросов `SELECT` ClickHouse анализирует возможность использования индекса. Индекс может использоваться, если в секции `WHERE/PREWHERE`, в качестве одного из элементов конъюнкции, или целиком, есть выражение, представляющее операции сравнения на равенства, неравенства, а также `IN` или `LIKE` с фиксированным префиксом, над столбцами или выражениями, входящими в первичный ключ или ключ партиционирования, либо над некоторыми частично монотонными функциями от этих столбцов, а также логические связки над такими выражениями.

Таким образом, обеспечивается возможность быстро выполнять запросы по одному или многим диапазонам первичного ключа. Например, в указанном примере будут быстро работать запросы для конкретного счётчика; для конкретного счётчика и диапазона дат; для конкретного счётчика и даты, для нескольких счётчиков и диапазона дат и т. п.

Рассмотрим движок сконфигурированный следующим образом:

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

В этом случае в запросах:

```
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 будет использовать индекс по первичному ключу для отсечения не подходящих данных, а также ключ партиционирования по месяцам для отсечения партиций, которые находятся в не подходящих диапазонах дат.

Запросы выше показывают, что индекс используется даже для сложных выражений. Чтение из таблицы организовано так, что использование индекса не может быть медленнее, чем `full scan`.

В примере ниже индекс не может использоваться.

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

Чтобы проверить, сможет ли ClickHouse использовать индекс при выполнении запроса, используйте настройки `force_index_by_date` и `force_primary_key`.

Ключ партиционирования по месяцам обеспечивает чтение только тех блоков данных, которые содержат даты из нужного диапазона. При этом блок данных может содержать данные за многие даты (до целого месяца). В пределах одного блока данные упорядочены по первичному ключу, который может не содержать дату в качестве первого столбца. В связи с этим, при использовании запроса с указанием условия только на дату, но не на префикс первичного ключа, будет читаться данных больше, чем за одну дату.

## Использование индекса для частично-монотонных первичных ключей

Рассмотрим, например, дни месяца. Они образуют последовательность [монотонную](#) в течение одного месяца, но не монотонную на более длительных периодах. Это частично-монотонная последовательность. Если пользователь создаёт таблицу с частично-монотонным первичным ключом, ClickHouse как обычно создаёт разреженный индекс. Когда пользователь выбирает данные из такого рода таблиц, ClickHouse анализирует условия запроса. Если пользователь хочет получить данные между двумя метками индекса, и обе эти метки находятся внутри одного месяца, ClickHouse может использовать индекс в данном конкретном случае, поскольку он может рассчитать расстояние между параметрами запроса и индексными метками.

ClickHouse не может использовать индекс, если значения первичного ключа в диапазоне параметров запроса не представляют собой монотонную последовательность. В этом случае ClickHouse использует метод полного сканирования.

ClickHouse использует эту логику не только для последовательностей дней месяца, но и для любого частично-монотонного первичного ключа.

## Индексы пропуска данных

Объявление индексов при определении столбцов в запросе CREATE.

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

Для таблиц семейства \*MergeTree можно задать дополнительные индексы в секции столбцов.

Индексы агрегируют для заданного выражения некоторые данные, а потом при `SELECT` запросе используют для пропуска блоков данных (пропускаемый блок состоит из гранул данных в количестве равном гранулярности данного индекса), на которых секция `WHERE` не может быть выполнена, тем самым уменьшая объём данных читаемых с диска.

## Пример

```
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()
...
```

Эти индексы смогут использоваться для оптимизации следующих запросов

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

## Доступные индексы

- `minmax` — хранит минимум и максимум выражения (если выражение - `Tuple`, то для каждого элемента `Tuple`), используя их для пропуска блоков аналогично первичному ключу.
- `set(max_rows)` — хранит уникальные значения выражения на блоке в количестве не более `max_rows` (если `max_rows = 0`, то ограничений нет), используя их для пропуска блоков, оценивая выполнимость `WHERE` выражения на хранимых данных.
- `ngrambf_v1(n, size_of_bloom_filter_in_bytes, number_of_hash_functions, random_seed)` — хранит **фильтр Блума**, содержащий все N-граммы блока данных. Работает только с данными форматов `String`, `FixedString` и `Map` с ключами типа `String` или `fixedString`. Может быть использован для оптимизации выражений `EQUALS`, `LIKE` и `IN`.
  - `n` — размер N-грамм,
  - `size_of_bloom_filter_in_bytes` — размер в байтах фильтра Блума (можно использовать большие значения, например, 256 или 512, поскольку сжатие компенсирует возможные издержки).
  - `number_of_hash_functions` — количество хеш-функций, использующихся в фильтре Блума.
  - `random_seed` — состояние генератора случайных чисел для хеш-функций фильтра Блума.
- `tokenbf_v1(size_of_bloom_filter_in_bytes, number_of_hash_functions, random_seed)` — то же, что `ngrambf_v1`, но хранит токены вместо N-грамм. Токены — это последовательности символов, разделенные не буквенно-цифровыми символами.
- `bloom_filter([false_positive])` — **фильтр Блума** для указанных столбцов.

Необязательный параметр `false_positive` — это вероятность получения ложноположительного срабатывания. Возможные значения: (0, 1). Значение по умолчанию: 0.025.

Поддерживаемые типы данных: `Int*`, `UInt*`, `Float*`, `Enum`, `Date`, `DateTime`, `String`, `FixedString`.

Фильтром могут пользоваться функции: `equals`, `notEquals`, `in`, `notin`, `has`.

## Примеры

```
INDEX b (u64 * length(str), i32 + f64 * 100, date, str) TYPE minmax GRANULARITY 4  
INDEX b (u64 * length(str), i32 + f64 * 100, date, str) TYPE set(100) GRANULARITY 4
```

## Поддержка для функций

Условия в секции `WHERE` содержат вызовы функций, оперирующих со столбцами. Если столбец - часть индекса, ClickHouse пытается использовать индекс при выполнении функции. Для разных видов индексов, ClickHouse поддерживает различные наборы функций, которые могут использоваться индексами.

Индекс `set` используется со всеми функциями. Наборы функций для остальных индексов представлены в таблице ниже.

Функция (оператор) / Индекс	primary key	minmax	ngrambf_v1	tokenbf_v1	bloom_filter
<code>equals (=, ==)</code>	✓	✓	✓	✓	✓

Функция (оператор) / Индекс	primary key	minmax	ngrambf_v1	tokenbf_v1	bloom_filter
notEquals(!=, \<>)	✓	✓	✓	✓	✓
like	✓	✓	✓	✓	✗
notLike	✓	✓	✓	✓	✗
startsWith	✓	✓	✓	✓	✗
endsWith	✗	✗	✓	✓	✗
multiSearchAny	✗	✗	✓	✗	✗
in	✓	✓	✓	✓	✓
notin	✓	✓	✓	✓	✓
less (\<)	✓	✓	✗	✗	✗
greater (>)	✓	✓	✗	✗	✗
lessOrEquals (\<=)	✓	✓	✗	✗	✗
greaterOrEquals (>=)	✓	✓	✗	✗	✗
empty	✓	✓	✗	✗	✗
notEmpty	✓	✓	✗	✗	✗
hasToken	✗	✗	✗	✓	✗

Функции с постоянным аргументом, который меньше, чем размер ngram не могут использовать индекс ngrambf\_v1 для оптимизации запроса.

Фильтры Блума могут иметь ложнопозитивные срабатывания, следовательно индексы ngrambf\_v1, tokenbf\_v1 и bloom\_filter невозможно использовать для оптимизации запросов, в которых результат функции предполагается false, например:

- Можно оптимизировать:
  - s LIKE '%test%'
  - NOT s NOT LIKE '%test%'
  - s = 1
  - NOT s != 1
  - startsWith(s, 'test')

- Нельзя оптимизировать:

- NOT s LIKE '%test%'
- s NOT LIKE '%test%'
- NOT s = 1
- s != 1
- NOT startsWith(s, 'test')

## Проекции

Проекции похожи на [материализованные представления](#), но определяются на уровне кусков данных. Это обеспечивает гарантии согласованности данных наряду с автоматическим использованием в запросах.

Проекции — это экспериментальная возможность. Чтобы включить поддержку проекций, установите настройку `allow_experimental_projection_optimization` в значение 1. См. также настройку `force_optimize_projection`.

Проекции не поддерживаются для запросов `SELECT` с модификатором `FINAL`.

### Запрос проекции

Запрос проекции — это то, что определяет проекцию. Такой запрос неявно выбирает данные из родительской таблицы.

#### Синтаксис

```
SELECT <column list expr> [GROUP BY] <group keys expr> [ORDER BY] <expr>
```

Проекции можно изменить или удалить с помощью запроса [ALTER](#).

### Хранение проекции

Проекции хранятся в каталоге куска данных. Это похоже на хранение индексов, но используется подкаталог, в котором хранится анонимный кусок таблицы `MergeTree`. Таблица создается запросом определения проекции.

Если присутствует секция `GROUP BY`, то используется движок [AggregatingMergeTree](#), а все агрегатные функции преобразуются в `AggregateFunction`.

Если присутствует секция `ORDER BY`, таблица `MergeTree` использует ее в качестве выражения для первичного ключа.

Во время процесса слияния кусок данных проекции объединяется с помощью процедуры слияния хранилища. Контрольная сумма куска данных родительской таблицы включает кусок данных проекции. Другие процедуры аналогичны индексам пропуска данных.

### Анализ запросов

1. Проверьте, можно ли использовать проекцию в данном запросе, то есть, что с ней получается тот же результат, что и с запросом к базовой таблице.
2. Выберите наиболее подходящее совпадение, содержащее наименьшее количество гранул для чтения.
3. План запроса, который использует проекции, отличается от того, который использует исходные куски данных. Если в некоторых кусках проекции отсутствуют, можно расширить план, чтобы «проецировать» на лету.

## Конкурентный доступ к данным

Для конкурентного доступа к таблице используется мультиверсионность. То есть, при одновременном чтении и обновлении таблицы, данные будут читаться из набора кусочков, актуального на момент запроса. Длинных блокировок нет. Вставки никак не мешают чтениям.

Чтения из таблицы автоматически распараллеливаются.

## TTL для столбцов и таблиц

Определяет время жизни значений.

Секция `TTL` может быть установлена как для всей таблицы, так и для каждого отдельного столбца. Для таблиц можно установить правила `TTL` для фонового перемещения кусков данных на целевые диски или тома, или правила повторного сжатия кусков данных.

Выражения должны возвращать тип `Date` или `DateTime`.

### Синтаксис

Для задания времени жизни столбца:

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

Чтобы задать `interval`, используйте операторы [интервала времени](#), например:

```
TTL date_time + INTERVAL 1 MONTH  
TTL date_time + INTERVAL 15 HOUR
```

## TTL столбца

Когда срок действия значений в столбце истечёт, ClickHouse заменит их значениями по умолчанию для типа данных столбца. Если срок действия всех значений столбцов в части данных истек, ClickHouse удаляет столбец из куска данных в файловой системе.

Секцию `TTL` нельзя использовать для ключевых столбцов.

### Примеры

Создание таблицы с `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;
```

Добавление `TTL` на колонку существующей таблицы:

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

Изменение `TTL` у колонки:

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

## TTL таблицы

Для таблицы можно задать одно выражение для устаревания данных, а также несколько выражений, при срабатывании которых данные будут перемещены на **некоторый диск или том**. Когда некоторые данные в таблице устаревают, ClickHouse удаляет все соответствующие строки. Операции перемещения или повторного сжатия данных выполняются только когда устаревают все данные в куске.

```
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) ...]] ]
```

За каждым **TTL** выражением может следовать тип действия, которое выполняется после достижения времени, соответствующего результату **TTL** выражения:

- **DELETE** - удалить данные (действие по умолчанию);
- **RECOMPRESS codec\_name** - повторно сжать данные с помощью кодека **codec\_name**;
- **TO DISK 'aaa'** - переместить данные на диск **aaa**;
- **TO VOLUME 'bbb'** - переместить данные на том **bbb**;
- **GROUP BY** - агрегировать данные.

В секции **WHERE** можно задать условие удаления или агрегирования устаревших строк (для перемещения и сжатия условие **WHERE** не применимо).

Колонки, по которым агрегируются данные в **GROUP BY**, должны являться префиксом первичного ключа таблицы.

Если колонка не является частью выражения **GROUP BY** и не задается напрямую в секции **SET**, в результирующих строках она будет содержать случайное значение, взятое из одной из структурированных строк (как будто к ней применяется агрегирующая функция **any**).

## Примеры

Создание таблицы с **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';
```

Изменение **TTL**:

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

Создание таблицы, в которой строки устаревают через месяц. Устаревшие строки удаляются, если дата выпадает на понедельник:

```
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;
```

Создание таблицы, в которой куски с устаревшими данными повторно сжимаются:

```
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;
```

Создание таблицы, где устаревшие строки агрегируются. В результирующих строках колонка `x` содержит максимальное значение по сгруппированным строкам, `y` — минимальное значение, а `d` — случайное значение из одной из сгруппированных строк.

```
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);
```

## Удаление устаревших данных

Данные с истекшим `TTL` удаляются, когда ClickHouse мёржит куски данных.

Когда ClickHouse видит, что некоторые данные устарели, он выполняет внеплановые мёржи. Для управление частотой подобных мёржей, можно задать настройку `merge_with_ttl_timeout`. Если её значение слишком низкое, придется выполнять много внеплановых мёржей, которые могут начать потреблять значительную долю ресурсов сервера.

Если вы выполните запрос `SELECT` между слияниями вы можете получить устаревшие данные. Чтобы избежать этого используйте запрос `OPTIMIZE` перед `SELECT`.

### См. также

- настройку `ttl_only_drop_parts`

# Хранение данных таблицы на нескольких блочных устройствах

## Введение

Движки таблиц семейства MergeTree могут хранить данные на нескольких блочных устройствах. Это может оказаться полезным, например, при неявном разделении данных одной таблицы на «горячие» и «холодные». Наиболее свежая часть занимает малый объём и запрашивается регулярно, а большой хвост исторических данных запрашивается редко. При наличии в системе нескольких дисков, «горячая» часть данных может быть размещена на быстрых дисках (например, на NVMe SSD или в памяти), а холодная на более медленных (например, HDD).

Минимальной перемещаемой единицей для MergeTree является кусок данных (data part). Данные одного куска могут находиться только на одном диске. Куски могут перемещаться между дисками в фоне, согласно пользовательским настройкам, а также с помощью запросов `ALTER`.

## Термины

- Диск — примонтированное в файловой системе блочное устройство.
- Диск по умолчанию — диск, на котором находится путь, указанный в конфигурационной настройке сервера `path`.
- Том (Volume) — упорядоченный набор равноценных дисков (схоже с `JBOD`)
- Политика хранения (StoragePolicy) — множество томов с правилами перемещения данных между ними.

У всех описанных сущностей при создании указываются имена, можно найти в системных таблицах `system.storage_policies` и `system.disks`. Имя политики хранения можно указать в настройке `storage_policy` движков таблиц семейства MergeTree.

## Конфигурация

Диски, тома и политики хранения задаются внутри тега `<storage_configuration>` в основном файле `config.xml` или в отдельном файле в директории `config.d`.

Структура конфигурации:

```
<storage_configuration>
  <disks>
    <disk_name_1> <!-- disk name -->
      <path>/mnt/fast_ssd/clickhouse/</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>
```

Теги:

- `<disk_name_N>` — имя диска. Имена должны быть разными для всех дисков.

- `path` — путь по которому будут храниться данные сервера (каталоги `data` и `shadow`), должен быть терминирован `/`.
- `keep_free_space_bytes` — размер зарезервированного свободного места на диске.

Порядок задания дисков не имеет значения.

Общий вид конфигурации политик хранения:

```
<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>
```

Тэги:

- `policy_name_N` — название политики. Названия политик должны быть уникальны.
- `volume_name_N` — название тома. Названия томов должны быть уникальны.
- `disk` — диск, находящийся внутри тома.
- `max_data_part_size_bytes` — максимальный размер куска данных, который может находиться на любом из дисков этого тома. Если в результате слияния размер куска ожидается больше, чем `max_data_part_size_bytes`, то этот кусок будет записан в следующий том. В основном эта функция позволяет хранить новые / мелкие куски на горячем (SSD) томе и перемещать их на холодный (HDD) том, когда они достигают большого размера. Не используйте этот параметр, если политика имеет только один том.
- `move_factor` — доля доступного свободного места на томе, если места становится меньше, то данные начнут перемещение на следующий том, если он есть (по умолчанию 0.1).
- `prefer_not_to_merge` — Отключает слияние кусков данных, хранящихся на данном томе. Если данная настройка включена, то слияние данных, хранящихся на данном томе, не допускается. Это позволяет контролировать работу ClickHouse с медленными дисками.

Примеры конфигураций:

```

<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>

```

В приведенном примере, политика `hdd_in_order` реализует принцип **round-robin**. Так как в политике есть всего один том (`single`), то все записи производятся на его диски по круговому циклу. Такая политика может быть полезна при наличии в системе нескольких похожих дисков, но при этом не сконфигурирован RAID. Учтите, что каждый отдельный диск ненадёжен и чтобы не потерять важные данные это необходимо скомпенсировать за счет хранения данных в трёх копиях.

Если система содержит диски различных типов, то может пригодиться политика `moving_from_ss_to_hdd`. В томе `hot` находится один SSD-диск (`fast_ss`), а также задается ограничение на максимальный размер куска, который может храниться на этом томе (1GB). Все куски такой таблицы больше 1GB будут записываться сразу на том `cold`, в котором содержится один HDD-диск `disk1`. Также, при заполнении диска `fast_ss` более чем на 80% данные будут переносится на диск `disk1` фоновым процессом.

Порядок томов в политиках хранения важен, при достижении условий на переполнение тома данные переносятся на следующий. Порядок дисков в томах так же важен, данные пишутся по очереди на каждый из них.

После задания конфигурации политик хранения их можно использовать, как настройку при создании таблиц:

```
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_ssds_to_hdd'
```

По умолчанию используется политика хранения default в которой есть один том и один диск, указанный в `<path>`.

Изменить политику хранения после создания таблицы можно при помощи запроса [ALTER TABLE ... MODIFY SETTING]. При этом необходимо учесть, что новая политика должна содержать все тома и диски предыдущей политики с теми же именами.

Количество потоков для фоновых перемещений кусков между дисками можно изменить с помощью настройки `background_move_pool_size`

## Особенности работы

В таблицах MergeTree данные попадают на диск несколькими способами:

- В результате вставки (запрос `INSERT`).
- В фоновых операциях слияний и **мутаций**.
- При скачивании данных с другой реплики.
- В результате заморозки партиций **ALTER TABLE ... FREEZE PARTITION**.

Во всех случаях, кроме мутаций и заморозки партиций, при записи куска выбирается том и диск в соответствии с указанной конфигурацией хранилища:

1. Выбирается первый по порядку том, на котором есть свободное место для записи куска (`unreserved_space > current_part_size`) и который позволяет записывать куски требуемого размера `max_data_part_size_bytes > current_part_size`.
2. Внутри тома выбирается следующий диск после того, на который была предыдущая запись и на котором свободного места больше чем размер куска (`unreserved_space - keep_free_space_bytes > current_part_size`)

Мутации и запросы заморозки партиций в реализации используют **жесткие ссылки**. Жесткие ссылки между различными дисками не поддерживаются, поэтому в случае таких операций куски размещаются на тех же дисках, что и исходные.

В фоне куски перемещаются между томами на основе информации о занятом месте (настройка `move_factor`) по порядку, в котором указаны тома в конфигурации. Данные никогда не перемещаются с последнего тома и на первый том. Следить за фоновыми перемещениями можно с помощью системных таблиц `system.part_log` (поле `type = MOVE_PART`) и `system.parts` (поля `path` и `disk`). Также подробная информация о перемещениях доступна в логах сервера.

С помощью запроса **ALTER TABLE ... MOVE PART|PARTITION ... TO VOLUME|DISK ...** пользователь может принудительно перенести кусок или партицию с одного раздела на другой. При этом учитываются все ограничения, указанные для фоновых операций. Запрос самостоятельно инициирует процесс перемещения не дожидаясь фоновых операций. В случае недостатка места или неудовлетворения ограничениям пользователь получит сообщение об ошибке.

Перемещения данных не взаимодействуют с репликацией данных, поэтому на разных репликах одной и той же таблицы могут быть указаны разные политики хранения.

После выполнения фоновых слияний или мутаций старые куски не удаляются сразу, а через некоторое время (табличная настройка `old_parts_lifetime`). Также они не перемещаются на другие тома или диски, поэтому до момента удаления они продолжают учитываться при подсчёте занятого дискового пространства.

## Использование сервиса S3 для хранения данных

Таблицы семейства MergeTree могут хранить данные в сервисе **S3** при использовании диска типа `s3`.

Конфигурация:

```
<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>
    <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>
```

Обязательные параметры:

- `endpoint` — URL точки приема запроса на стороне S3 в **форматах path** или **virtual hosted**. URL точки должен содержать бакет и путь к корневой директории на сервере, где хранятся данные.
- `access_key_id` — id ключа доступа к S3.
- `secret_access_key` — секретный ключ доступа к S3.

Необязательные параметры:

- `region` — название региона S3.
- `use_environment_credentials` — признак, нужно ли считывать учетные данные AWS из сетевого окружения, а также из переменных окружения `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY` и `AWS_SESSION_TOKEN`, если они есть. Значение по умолчанию: `false`.
- `use_insecure_imds_request` — признак, нужно ли использовать менее безопасное соединение при выполнении запроса к IMDS при получении учётных данных из метаданных Amazon EC2. Значение по умолчанию: `false`.
- `proxy` — конфигурация прокси-сервера для конечной точки S3. Каждый элемент `uri` внутри блока `proxy` должен содержать URL прокси-сервера.
- `connect_timeout_ms` — таймаут подключения к сокету в миллисекундах. Значение по умолчанию: 10 секунд.

- `request_timeout_ms` — таймаут выполнения запроса в миллисекундах. Значение по умолчанию: 5 секунд.
- `retry_attempts` — число попыток выполнения запроса в случае возникновения ошибки. Значение по умолчанию: 10.
- `single_read_retries` — число попыток выполнения запроса в случае возникновения ошибки в процессе чтения. Значение по умолчанию: 4.
- `min_bytes_for_seek` — минимальное количество байтов, которые используются для операций поиска вместо последовательного чтения. Значение по умолчанию: 1 МБайт.
- `metadata_path` — путь к локальному файловому хранилищу для хранения файлов с метаданными для S3. Значение по умолчанию: `/var/lib/clickhouse/disks/<disk_name>/`.
- `cache_enabled` — признак, разрешено ли хранение кэша засечек и файлов индекса в локальной файловой системе. Значение по умолчанию: `true`.
- `cache_path` — путь в локальной файловой системе, где будут храниться кэш засечек и файлы индекса. Значение по умолчанию: `/var/lib/clickhouse/disks/<disk_name>/cache/`.
- `skip_access_check` — признак, выполнять ли проверку доступов при запуске диска. Если установлено значение `true`, то проверка не выполняется. Значение по умолчанию: `false`.

Диск S3 может быть сконфигурирован как `main` или `cold`:

```
<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>
```

Если диск сконфигурирован как `cold`, данные будут переноситься в S3 при срабатывании правил TTL или когда свободное место на локальном диске станет меньше порогового значения, которое определяется как `move_factor * disk_size`.

## Репликация данных

Репликация поддерживается только для таблиц семейства MergeTree:

- ReplicatedMergeTree
- ReplicatedSummingMergeTree
- ReplicatedReplacingMergeTree
- ReplicatedAggregatingMergeTree
- ReplicatedCollapsingMergeTree
- ReplicatedVersionedCollapsingMergeTree
- ReplicatedGraphiteMergeTree

Репликация работает на уровне отдельных таблиц, а не всего сервера. То есть, на сервере могут быть расположены одновременно реплицируемые и не реплицируемые таблицы.

Репликация не зависит от шардирования. На каждом шарде репликация работает независимо.

Реплицируются сжатые данные запросов `INSERT`, `ALTER` (см. подробности в описании запроса [ALTER](#)).

Запросы `CREATE`, `DROP`, `ATTACH`, `DETACH` и `RENAME` выполняются на одном сервере и не реплицируются:

- Запрос `CREATE TABLE` создаёт новую реплицируемую таблицу на том сервере, где его выполнили. Если таблица уже существует на других серверах, запрос добавляет новую реплику.
- `DROP TABLE` удаляет реплику, расположенную на том сервере, где выполняется запрос.
- Запрос `RENAME` переименовывает таблицу на одной реплик. Другими словами, реплицируемые таблицы на разных репликах могут называться по-разному.

ClickHouse хранит метаинформацию о репликах в [Apache ZooKeeper](#). Используйте ZooKeeper 3.4.5 или новее.

Для использования репликации, установите параметры в секции `zookeeper` конфигурации сервера.

## Внимание

Не пренебрегайте настройками безопасности. ClickHouse поддерживает [ACL схему digest](#) подсистемы безопасности ZooKeeper.

Пример указания адресов кластера ZooKeeper:

```
<zookeeper>
  <node index="1">
    <host>example1</host>
    <port>2181</port>
  </node>
  <node index="2">
    <host>example2</host>
    <port>2181</port>
  </node>
  <node index="3">
    <host>example3</host>
    <port>2181</port>
  </node>
</zookeeper>
```

Можно указать любой имеющийся у вас ZooKeeper-кластер - система будет использовать в нём одну директорию для своих данных (директория указывается при создании реплицируемой таблицы).

Если в конфигурационном файле не настроен ZooKeeper, то вы не сможете создать реплицируемые таблицы, а уже имеющиеся реплицируемые таблицы будут доступны в режиме только на чтение.

При запросах `SELECT`, ZooKeeper не используется, т.е. репликация не влияет на производительность `SELECT` и запросы работают так же быстро, как и для нереплицируемых таблиц. При запросах к распределенным реплицированным таблицам поведение ClickHouse регулируется настройками `max_replica_delay_for_distributed_queries` и `fallback_to_stale_replicas_for_distributed_queries`.

При каждом запросе `INSERT`, делается около десятка записей в ZooKeeper в рамках нескольких транзакций. (Чтобы быть более точным, это для каждого вставленного блока данных; запрос `INSERT` содержит один блок или один блок на `max_insert_block_size = 1048576` строк.) Это приводит к некоторому увеличению задержек при `INSERT`, по сравнению с нереплицируемыми таблицами. Но если придерживаться обычных рекомендаций - вставлять данные пачками не более одного `INSERT` в секунду, то это не составляет проблем. На всём кластере ClickHouse, использующим для координации один кластер ZooKeeper, может быть в совокупности несколько сотен `INSERT` в секунду. Пропускная способность при вставке данных (количество строчек в секунду) такая же высокая, как для нереплицируемых таблиц.

Для очень больших кластеров, можно использовать разные кластеры ZooKeeper для разных шардов. Впрочем, на кластере Яндекс.Метрики (примерно 300 серверов) такой необходимости не возникает.

Репликация асинхронная, мульти-мастер. Запросы `INSERT` и `ALTER` можно направлять на любой доступный сервер. Данные вставляются на сервер, где выполнен запрос, а затем скопируются на остальные серверы. В связи с асинхронностью, только что вставленные данные появляются на остальных репликах с небольшой задержкой. Если часть реплик недоступна, данные на них запишутся тогда, когда они станут доступны. Если реплика доступна, то задержка составляет столько времени, сколько требуется для передачи блока сжатых данных по сети. Количество потоков для выполнения фоновых задач можно задать с помощью настройки `background_schedule_pool_size`.

Движок `ReplicatedMergeTree` использует отдельный пул потоков для скачивания кусков данных. Размер пула ограничен настройкой `background_fetches_pool_size`, которую можно указать при перезапуске сервера.

По умолчанию, запрос `INSERT` ждёт подтверждения записи только от одной реплики. Если данные были успешно записаны только на одну реплику, и сервер с этой репликой перестал существовать, то записанные данные будут потеряны. Вы можете включить подтверждение записи от нескольких реплик, используя настройку `insert_quorum`.

Каждый блок данных записывается атомарно. Запрос `INSERT` разбивается на блоки данных размером до `max_insert_block_size = 1048576` строк. То есть, если в запросе `INSERT` менее 1048576 строк, то он делается атомарно.

Блоки данных дедуплицируются. При многократной записи одного и того же блока данных (блоков данных одинакового размера, содержащих одни и те же строчки в одном и том же порядке), блок будет записан только один раз. Это сделано для того, чтобы в случае сбоя в сети, когда клиентское приложение не может понять, были ли данные записаны в БД, можно было просто повторить запрос `INSERT`. При этом не имеет значения, на какую реплику будут отправлены `INSERT`-ы с одинаковыми данными. Запрос `INSERT` идемпотентный. Параметры дедуплицирования регулируются настройками сервера `merge_tree`

При репликации, по сети передаются только исходные вставляемые данные. Дальнейшие преобразования данных (слияния) координируются и делаются на всех репликах одинаковым образом. За счёт этого минимизируется использование сети, и благодаря этому, репликация хорошо работает при расположении реплик в разных data-центрах. (Стоит заметить, что дублирование данных в разных data-центрах, по сути, является основной задачей репликации).

Количество реплик одних и тех же данных может быть произвольным. В Яндекс.Метрике в продакшне используется двукратная репликация. На каждом сервере используется RAID-5 или RAID-6, в некоторых случаях RAID-10. Это является сравнительно надёжным и удобным для эксплуатации решением.

Система следит за синхронностью данных на репликах и умеет восстанавливаться после сбоя. Восстановление после сбоя автоматическое (в случае небольших различий в данных) или полуавтоматическое (когда данные отличаются слишком сильно, что может свидетельствовать об ошибке конфигурации).

## Создание реплицируемых таблиц

В начало имени движка таблицы добавляется `Replicated`. Например, `ReplicatedMergeTree`.

### Параметры `Replicated*MergeTree`

- `zoo_path` — путь к таблице в ZooKeeper.
- `replica_name` — имя реплики в ZooKeeper.
- `other_parameters` — параметры движка, для которого создаётся реплицированная версия, например, версия для `ReplacingMergeTree`.

Пример:

```
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);
```

### ▶ Пример в устаревшем синтаксисе

Как видно в примере, эти параметры могут содержать подстановки в фигурных скобках. Эти подстановки заменяются на соответствующие значения из конфигурационного файла, из секции `macros`.

Пример:

```
<macros>
<layer>05</layer>
<shard>02</shard>
<replica>example05-02-1.yandex.ru</replica>
</macros>
```

Путь к таблице в ZooKeeper должен быть разным для каждой реплицируемой таблицы. В том числе, для таблиц на разных шардах, должны быть разные пути.

В данном случае, путь состоит из следующих частей:

`/clickhouse/tables/` — общий префикс. Рекомендуется использовать именно его.

`{layer}-{shard}` — идентификатор шарда. В данном примере он состоит из двух частей, так как на кластере Яндекс.Метрики используется двухуровневое шардирование. Для большинства задач, оставьте только подстановку `{shard}`, которая будет раскрываться в идентификатор шарда.

`table_name` - имя узла для таблицы в ZooKeeper. Разумно делать его таким же, как имя таблицы. Оно указывается явно, так как, в отличие от имени таблицы, оно не меняется после запроса `RENAME`.  
Подсказка: можно также указать имя базы данных перед `table_name`, например `db_name.table_name`

Можно использовать две встроенных подстановки `{database}` и `{table}`, они раскрываются в имя таблицы и в имя базы данных соответственно (если эти подстановки не переопределены в секции `macros`). Т.о. Zookeeper путь можно задать как `'/clickhouse/tables/{layer}-{shard}/{database}/{table}'`.  
Будьте осторожны с переименованиями таблицы при использовании этих автоматических подстановок. Путь в Zookeeper-е нельзя изменить, а подстановка при переименовании таблицы раскроется в другой путь, таблица будет обращаться к несуществующему в Zookeeper-е пути и перейдет в режим только для чтения.

Имя реплики — то, что идентифицирует разные реплики одной и той же таблицы. Можно использовать для него имя сервера, как показано в примере. Впрочем, достаточно, чтобы имя было уникально лишь в пределах каждого шарда.

Можно не использовать подстановки, а указать соответствующие параметры явно. Это может быть удобным для тестирования и при настройке маленьких кластеров. Однако в этом случае нельзя пользоваться распределенными DDL-запросами (`ON CLUSTER`).

При работе с большими кластерами мы рекомендуем использовать подстановки, они уменьшают вероятность ошибки.

Можно указать аргументы по умолчанию для движка реплицируемых таблиц в файле конфигурации сервера.

```
<default_replica_path>/clickhouse/tables/{shard}/{database}/{table}</default_replica_path>
<default_replica_name>{replica}</default_replica_name>
```

В этом случае можно опустить аргументы при создании таблиц:

```
CREATE TABLE table_name (
    x UInt32
) ENGINE = ReplicatedMergeTree
ORDER BY x;
```

Это будет эквивалентно следующему запросу:

```
CREATE TABLE table_name (
    x UInt32
) ENGINE = ReplicatedMergeTree('/clickhouse/tables/{shard}/{database}/table_name', '{replica}')
ORDER BY x;
```

Выполните запрос `CREATE TABLE` на каждой реплике. Запрос создаёт новую реплицируемую таблицу, или добавляет новую реплику к имеющимся.

Если вы добавляете новую реплику после того, как таблица на других репликах уже содержит некоторые данные, то после выполнения запроса, данные на новую реплику будут скачаны с других реплик. То есть, новая реплика синхронизирует себя с остальными.

Для удаления реплики, выполните запрос `DROP TABLE`. При этом, удаляется только одна реплика — расположенная на том сервере, где вы выполняете запрос.

## Восстановление после сбоя

Если при старте сервера, недоступен ZooKeeper, реплицируемые таблицы переходят в режим только для чтения. Система будет пытаться периодически установить соединение с ZooKeeper.

Если при `INSERT` недоступен ZooKeeper, или происходит ошибка при взаимодействии с ним, будет выкинуто исключение.

При подключении к ZooKeeper, система проверяет соответствие между имеющимся в локальной файловой системе набором данных и ожидаемым набором данных (информация о котором хранится в ZooKeeper). Если имеются небольшие несоответствия, то система устраняет их, синхронизируя данные с реплик.

Обнаруженные битые куски данных (с файлами несоответствующего размера) или неизвестные куски (куски, записанные в файловую систему, но информация о которых не была записана в ZooKeeper) переносятся в поддиректорию `detached` (не удаляются). Недостающие куски скачиваются с реплик.

Стоит заметить, что ClickHouse не делает самостоятельно никаких деструктивных действий типа автоматического удаления большого количества данных.

При старте сервера (или создании новой сессии с ZooKeeper), проверяется только количество и размеры всех файлов. Если у файлов совпадают размеры, но изменены байты где-то посередине, то это обнаруживается не сразу, а только при попытке их прочитать при каком-либо запросе `SELECT`. Запрос кинет исключение о несоответствующей чекsumме или размере сжатого блока. В этом случае, куски данных добавляются в очередь на проверку, и при необходимости, скачиваются с реплик.

Если обнаруживается, что локальный набор данных слишком сильно отличается от ожидаемого, то срабатывает защитный механизм. Сервер сообщает об этом в лог и отказывается запускаться. Это сделано, так как такой случай может свидетельствовать об ошибке конфигурации - например, если реплика одного шарда была случайно сконфигурирована, как реплика другого шарда. Тем не менее, пороги защитного механизма поставлены довольно низкими, и такая ситуация может возникнуть и при обычном восстановлении после сбоя. В этом случае, восстановление делается полуавтоматически - «по кнопке».

Для запуска восстановления, создайте в ZooKeeper узел `/path_to_table/replica_name/flags/force_restore_data` с любым содержимым или выполните команду для восстановления всех реплицируемых таблиц:

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

Затем запустите сервер. При старте, сервер удалит эти флаги и запустит восстановление.

## Восстановление в случае потери всех данных

Если на одном из серверов исчезли все данные и метаданные, восстановление делается следующим образом:

1. Установите на сервер ClickHouse. Корректно пропишите подстановки в конфигурационном файле, отвечающие за идентификатор шарда и реплики, если вы их используете.
2. Если у вас были нереплицируемые таблицы, которые должны быть вручную продублированы на серверах, скопируйте их данные (в директории `/var/lib/clickhouse/data/db_name/table_name/`) с реплики.

3. Скопируйте с реплики определения таблиц, находящиеся в `/var/lib/clickhouse/metadata/`. Если в определениях таблиц, идентификатор шарда или реплики, прописаны в явном виде - исправьте их, чтобы они соответствовали данной реплике. (Альтернативный вариант - запустить сервер и сделать самостоятельно все запросы `ATTACH TABLE`, которые должны были бы быть в соответствующих `.sql` файлах в `/var/lib/clickhouse/metadata/`.)
4. Создайте в ZooKeeper узел `/path_to_table/replica_name flags/force_restore_data` с любым содержимым или выполните команду для восстановления всех реплицируемых таблиц: `sudo -u clickhouse touch /var/lib/clickhouse/flags/force_restore_data`

Затем запустите сервер (перезапустите, если уже запущен). Данные будут скачаны с реплик.

В качестве альтернативного варианта восстановления, вы можете удалить из ZooKeeper информацию о потерянной реплике (`/path_to_table/replica_name`), и затем создать реплику заново, как написано в разделе [Создание реплицированных таблиц](#).

Отсутствует ограничение на использование сетевой полосы при восстановлении. Имейте это ввиду, если восстанавливаете сразу много реплик.

## Преобразование из MergeTree в ReplicatedMergeTree

Здесь и далее, под `MergeTree` подразумеваются все движки таблиц семейства `MergeTree`, так же для `ReplicatedMergeTree`.

Если у вас была таблица типа `MergeTree`, репликация которой делалась вручную, вы можете преобразовать её в реплицируемую таблицу. Это может понадобиться лишь в случаях, когда вы уже успели накопить большое количество данных в таблице типа `MergeTree`, а сейчас хотите включить репликацию.

Если на разных репликах данные отличаются, то сначала синхронизируйте их, либо удалите эти данные на всех репликах кроме одной.

Переименуйте имеющуюся `MergeTree` таблицу, затем создайте со старым именем таблицу типа `ReplicatedMergeTree`.

Перенесите данные из старой таблицы в поддиректорию `detached` в директории с данными новой таблицы (`/var/lib/clickhouse/data/db_name/table_name/`).

Затем добавьте эти куски данных в рабочий набор с помощью выполнения запросов `ALTER TABLE ATTACH PARTITION` на одной из реплик.

## Преобразование из ReplicatedMergeTree в MergeTree

Создайте таблицу типа `MergeTree` с другим именем. Перенесите в её директорию с данными все данные из директории с данными таблицы типа `ReplicatedMergeTree`. Затем удалите таблицу типа `ReplicatedMergeTree` и перезапустите сервер.

Если вы хотите избавиться от таблицы `ReplicatedMergeTree`, не запуская сервер, то

- удалите соответствующий файл `.sql` в директории с метаданными (`/var/lib/clickhouse/metadata/`);
- удалите соответствующий путь в ZooKeeper (`/path_to_table/replica_name`);

После этого, вы можете запустить сервер, создать таблицу типа `MergeTree`, перенести данные в её директорию, и перезапустить сервер.

## Восстановление в случае потери или повреждения метаданных на ZooKeeper кластере

Если данные в ZooKeeper оказались утеряны или повреждены, то вы можете сохранить данные, переместив их в нереплицируемую таблицу, как описано в пункте выше.

## Смотрите также

- [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](#)

## Произвольный ключpartitionирования

Partitionирование данных доступно для таблиц семейства [MergeTree](#) (включая [реплицированные таблицы](#)). Таблицы [MaterializedView](#), созданные на основе таблиц MergeTree, также поддерживают partitionирование.

Партиция – это набор записей в таблице, объединенных по какому-либо критерию. Например, партиция может быть по месяцу, по дню или по типу события. Данные для разных партиций хранятся отдельно. Это позволяет оптимизировать работу с данными, так как при обработке запросов будет использоваться только необходимое подмножество из всевозможных данных. Например, при получении данных за определенный месяц, ClickHouse будет считывать данные только за этот месяц.

Ключ partitionирования задается при [создании таблицы](#), в секции `PARTITION BY expr`. Ключ может представлять собой произвольное выражение из столбцов таблицы. Например, чтобы задать partitionирования по месяцам, можно использовать выражение `toYYYYMM(date_column)`:

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

Ключом partitionирования также может быть кортеж из выражений (аналогично [первичному ключу](#)). Например:

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

В этом примере задано partitionирование по типам событий, произошедших в течение текущей недели.

По умолчанию, ключ partitionирования с плавающей запятой не поддерживается. Чтобы использовать его, включите настройку [allow\\_floating\\_point\\_partition\\_key](#).

Каждая партиция состоит из отдельных фрагментов или так называемых [кусков данных](#). Каждый кусок отсортирован по первичному ключу. При вставке данных в таблицу каждая отдельная запись сохраняется в виде отдельного куска. Через некоторое время после вставки (обычно до 10 минут), ClickHouse выполняет в фоновом режиме слияние данных — в результате куски для одной и той же партии будут объединены в более крупный кусок.

## Info

Не рекомендуется делать слишком гранулированное партиционирование – то есть задавать партиции по столбцу, в котором будет слишком большой разброс значений (речь идет о порядке более тысячи партиций). Это приведет к скоплению большого числа файлов и файловых дескрипторов в системе, что может значительно снизить производительность запросов `SELECT`.

Чтобы получить набор кусков и партиций таблицы, можно воспользоваться системной таблицей `system.parts`. В качестве примера рассмотрим таблицу `visits`, в которой задано партиционирование по месяцам. Выполним `SELECT` для таблицы `system.parts`:

```
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

Столбец `partition` содержит имена всех партиций таблицы. Таблица `visits` из нашего примера содержит две партиции: `201901` и `201902`. Используйте значения из этого столбца в запросах `ALTER ... PARTITION`.

Столбец `name` содержит названия кусков партиций. Значения из этого столбца можно использовать в запросах `ALTER ATTACH PART`.

Столбец `active` отображает состояние куска. `1` означает, что кусок активен; `0` – неактивен. К неактивным можно отнести куски, оставшиеся после слияния данных. Поврежденные куски также отображаются как неактивные. Неактивные куски удаляются приблизительно через 10 минут после того, как было выполнено слияние.

Рассмотрим детальнее имя первого куска `201901_1_3_1`:

- `201901` имя партиции;
- `1` – минимальный номер блока данных;
- `3` – максимальный номер блока данных;
- `1` – уровень куска (глубина дерева слияний, которыми этот кусок образован).

## Info

Названия кусков для таблиц старого типа образуются следующим образом:

`20190117_20190123_2_2_0` (минимальная дата \_ максимальная дата \_ номер минимального блока \_ номер максимального блока \_ уровень).

Как видно из примера выше, таблица содержит несколько отдельных кусков для одной и той же партиции (например, куски 201901\_1\_3\_1 и 201901\_1\_9\_2 принадлежат партиции 201901). Это означает, что эти куски еще не были объединены – в файловой системе они хранятся отдельно. После того как будет выполнено автоматическое слияние данных (выполняется примерно спустя 10 минут после вставки данных), исходные куски будут объединены в один более крупный кусок и помечены как неактивные.

Вы можете запустить внеочередное слияние данных с помощью запроса **OPTIMIZE**. Пример:

```
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

Неактивные куски будут удалены примерно через 10 минут после слияния.

Другой способ посмотреть набор кусков и партиций – зайти в директорию с данными таблицы: `/var/lib/clickhouse/data/<database>/<table>/`. Например:

```
/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
```

'201901\_1\_1\_0', '201901\_1\_7\_1' и т. д. – это директории кусков партиции. Каждый кусок содержит данные только для соответствующего месяца (таблица в данном примере содержит партиционирование по месяцам).

Директория `detached` содержит куски, отсоединенные от таблицы с помощью запроса **DETACH**. Поврежденные куски также попадают в эту директорию – они не удаляются с сервера.

Сервер не использует куски из директории `detached`. Вы можете в любое время добавлять, удалять, модифицировать данные в директории `detached` – сервер не будет об этом знать, пока вы не сделаете запрос **ATTACH**.

Следует иметь в виду, что при работающем сервере нельзя вручную изменять набор кусков на файловой системе, так как сервер не будет знать об этом.

Для нереплицируемых таблиц, вы можете это делать при остановленном сервере, однако это не рекомендуется.

Для реплицируемых таблиц, набор кусков нельзя менять в любом случае.

ClickHouse позволяет производить различные манипуляции с кусками: удалять, копировать из одной таблицы в другую или создавать их резервные копии. Подробнее см. в разделе [Манипуляции сパーティциями и кусками](#).

# ReplacingMergeTree

Движок отличается от [MergeTree](#) тем, что выполняет удаление дублирующихся записей с одинаковым значением [ключа сортировки](#) (секция `ORDER BY`, не `PRIMARY KEY`).

Дедупликация данных производится лишь во время слияний. Слияние происходит в фоне в неизвестный момент времени, на который вы не можете ориентироваться. Некоторая часть данных может остаться необработанной. Хотя вы можете вызвать внеочередное слияние с помощью запроса `OPTIMIZE`, на это не стоит рассчитывать, так как запрос `OPTIMIZE` приводит к чтению и записи большого объёма данных.

Таким образом, `ReplacingMergeTree` подходит для фоновой чистки дублирующихся данных в целях экономии места, но не даёт гарантии отсутствия дубликатов.

## Создание таблицы

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
(
    name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1],
    name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2],
    ...
) ENGINE = ReplacingMergeTree([ver])
[PARTITION BY expr]
[ORDER BY expr]
[SAMPLE BY expr]
[SETTINGS name=value, ...]
```

Описание параметров запроса смотрите в [описании запроса](#).

### Внимание

Уникальность строк определяется `ORDER BY` секцией таблицы, а не `PRIMARY KEY`.

### Параметры `ReplacingMergeTree`

- `ver` — столбец с номером версии. Тип `UInt*`, `Date`, `DateTime` или `DateTime64`. Необязательный параметр.

При слиянии `ReplacingMergeTree` оставляет только строку для каждого уникального ключа сортировки:

- Последнюю в выборке, если `ver` не задан. Под выборкой здесь понимается набор строк в наборе кусков данных, участвующих в слиянии. Последний по времени создания кусок (последняя вставка) будет последним в выборке. Таким образом, после дедупликации для каждого значения ключа сортировки останется самая последняя строка из самой последней вставки.
- С максимальной версией, если `ver` задан.

### Секции запроса

При создании таблицы `ReplacingMergeTree` используются те же [секции](#), что и при создании таблицы `MergeTree`.

▶ [Устаревший способ создания таблицы](#)

# SummingMergeTree

Движок наследует функциональность [MergeTree](#). Отличие заключается в том, что для таблиц `SummingMergeTree` при слиянии кусков данных ClickHouse все строки с одинаковым первичным ключом (точнее, с одинаковым [ключом сортировки](#)) заменяет на одну, которая хранит только суммы значений из столбцов с цифровым типом данных. Если ключ сортировки подобран таким образом, что одному значению ключа соответствует много строк, это значительно уменьшает объём хранения и ускоряет последующую выборку данных.

Мы рекомендуем использовать движок в паре с `MergeTree`. В `MergeTree` храните полные данные, а `SummingMergeTree` используйте для хранения агрегированных данных, например, при подготовке отчетов. Такой подход позволит не утратить ценные данные из-за неправильно выбранного первичного ключа.

## Создание таблицы

```
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, ...]
```

Описание параметров запроса смотрите в [описании запроса](#).

### Параметры `SummingMergeTree`

- `columns` — кортеж с именами столбцов, в которых будут суммироваться данные. Необязательный параметр.

Столбцы должны иметь числовой тип и не должны входить в первичный ключ.

Если `columns` не задан, то ClickHouse суммирует значения во всех столбцах с числовым типом данных, не входящих в первичный ключ.

### Секции запроса

При создании таблицы `SummingMergeTree` используются те же [секции](#) запроса, что и при создании таблицы `MergeTree`.

#### ► Устаревший способ создания таблицы

## Пример использования

Рассмотрим следующую таблицу:

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

Добавим в неё данные:

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

ClickHouse может не полностью просуммировать все строки ([смотрите ниже по тексту](#)), поэтому при запросе мы используем агрегатную функцию `sum` и секцию `GROUP BY`.

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

key	sum(value)
2	1
1	3

## Обработка данных

При вставке данных в таблицу они сохраняются как есть. Периодически ClickHouse выполняет слияние вставленных кусков данных и именно в этот момент производится суммирование и замена многих строк с одинаковым первичным ключом на одну для каждого результирующего куска данных.

ClickHouse может слить куски данных таким образом, что не все строки с одинаковым первичным ключом окажутся в одном финальном куске, т.е. суммирование будет не полным. Поэтому, при выборке данных (`SELECT`) необходимо использовать агрегатную функцию `sum()` и секцию `GROUP BY` как описано в примере выше.

## Общие правила суммирования

Суммируются значения в столбцах с числовым типом данных. Набор столбцов определяется параметром `columns`.

Если значения во всех столбцах для суммирования оказались нулевыми, то строчка удаляется.

Для столбцов, не входящих в первичный ключ и не суммирующихся, выбирается произвольное значение из имеющихся.

Значения для столбцов, входящих в первичный ключ, не суммируются.

## Суммирование в столбцах `AggregateFunction`

Для столбцов типа `AggregateFunction` ClickHouse выполняет агрегацию согласно заданной функции, повторяя поведение движка `AggregatingMergeTree`.

## Вложенные структуры

Таблица может иметь вложенные структуры данных, которые обрабатываются особым образом.

Если название вложенной таблицы заканчивается на `Map` и она содержит не менее двух столбцов, удовлетворяющих критериям:

- первый столбец - числовой (`*Int*`, `Date`, `DateTime`) или строковый (`String`, `FixedString`), назовем его условно `key`,
- остальные столбцы - арифметические (`*Int*`, `Float32/64`), условно `(values...)`,

то вложенная таблица воспринимается как отображение `key => (values...)` и при слиянии её строк выполняется слияние элементов двух множеств по `key` со сложением соответствующих `(values...)`.

Примеры:

```
[(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)]
```

При запросе данных используйте функцию `sumMap(key, value)` для агрегации `Map`.

Для вложенной структуры данных не нужно указывать её столбцы в кортеже столбцов для суммирования.

## AggregatingMergeTree

Движок наследует функциональность `MergeTree`, изменяя логику слияния кусков данных. Все строки с одинаковым первичным ключом (точнее, с одинаковым `ключом сортировки`) ClickHouse заменяет на одну (в пределах одного куска данных), которая хранит объединение состояний агрегатных функций.

Таблицы типа `AggregatingMergeTree` могут использоваться для инкрементальной агрегации данных, в том числе, для агрегирующих материализованных представлений.

Движок обрабатывает все столбцы типа `AggregateFunction`.

Использование `AggregatingMergeTree` оправдано только в том случае, когда это уменьшает количество строк на порядки.

## Создание таблицы

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
(
    name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1],
    name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2],
    ...
) ENGINE = AggregatingMergeTree()
[PARTITION BY expr]
[ORDER BY expr]
[SAMPLE BY expr]
[SETTINGS name=value, ...]
```

Описание параметров запроса смотрите в [описании запроса](#).

### Секции запроса

При создании таблицы `AggregatingMergeTree` используются те же `секции`, что и при создании таблицы `MergeTree`.

#### ► Устаревший способ создания таблицы

## SELECT/INSERT данных

Для вставки данных используйте `INSERT SELECT` с агрегатными `-State`-функциями.

При выборке данных из таблицы `AggregatingMergeTree`, используйте `GROUP BY` и те же агрегатные функции, что и при вставке данных, но с суффиксом `-Merge`.

В запросах `SELECT` значения типа `AggregateFunction` выводятся во всех форматах, которые поддерживает ClickHouse, в виде implementation-specific бинарных данных. Если с помощью `SELECT` выполнить дамп данных, например, в формат `TabSeparated`, то потом этот дамп можно загрузить обратно с помощью запроса `INSERT`.

# Пример агрегирующего материализованного представления

Создаём материализованное представление типа AggregatingMergeTree, следящее за таблицей test.visits:

```
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;
```

Вставляем данные в таблицу test.visits:

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

Данные окажутся и в таблице и в представлении test.basic, которое выполнит агрегацию.

Чтобы получить агрегированные данные, выполним запрос вида SELECT ... GROUP BY ... из представления test.basic:

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

## CollapsingMergeTree

Движок наследует функциональность от MergeTree и добавляет в алгоритм слияния кусков данных логику сворачивания (удаления) строк.

CollapsingMergeTree асинхронно удаляет (сворачивает) пары строк, если все поля в ключе сортировки (ORDER BY) эквивалентны, за исключением специального поля Sign, которое может принимать значения 1 и -1. Строки без пары сохраняются. Подробнее смотрите в разделе [Сворачивание \(удаление\) строк](#).

Движок может значительно уменьшить объём хранения и, как следствие, повысить эффективность запросов SELECT.

## Создание таблицы

```
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, ...]
```

Подробности про CREATE TABLE смотрите в [описании запроса](#).

## Параметры CollapsingMergeTree

- sign — Имя столбца с типом строки: 1 — строка состояния, -1 — строка отмены состояния.

Тип данных столбца — `Int8`.

## Секции запроса

При создании таблицы с движком CollapsingMergeTree используются те же [секции запроса](#) что и при создании таблицы с движком MergeTree.

### ► Устаревший способ создания таблицы

## Сворачивание (удаление) строк

### Данные

Рассмотрим ситуацию, когда необходимо сохранять постоянно изменяющиеся данные для какого-либо объекта. Кажется логичным иметь одну строку для объекта и обновлять её при любом изменении, однако операция обновления является дорогостоящей и медленной для СУБД, поскольку требует перезаписи данных в хранилище. Если необходимо быстро записать данные, обновление не допустимо, но можно записать изменения объекта последовательно как описано ниже.

Используйте специальный столбец `Sign`. Если `Sign = 1`, то это означает, что строка является состоянием объекта, назовём её строкой состояния. Если `Sign = -1`, то это означает отмену состояния объекта с теми же атрибутами, назовём её строкой отмены состояния.

Например, мы хотим рассчитать, сколько страниц проверили пользователи на каком-то сайте и как долго они там находились. В какой-то момент времени мы пишем следующую строку с состоянием действий пользователя:

UserID	PageViews	Duration	Sign
4324182021466249494	5	146	1

Через некоторое время мы регистрируем изменение активности пользователя и записываем его следующими двумя строками.

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

Первая строка отменяет предыдущее состояние объекта (пользователя). Она должен повторять все поля из ключа сортировки для отменённого состояния за исключением `Sign`.

Вторая строка содержит текущее состояние.

Поскольку нам нужно только последнее состояние активности пользователя, строки

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

можно удалить, сворачивая (удаляя) устаревшее состояние объекта. CollapsingMergeTree выполняет это при слиянии кусков данных.

Зачем нужны две строки для каждого изменения описано в разделе [Алгоритм](#).

## Особенности подхода

1. Программа, которая записывает данные, должна помнить состояние объекта, чтобы иметь возможность отменить его. Стока отмены состояния должна содержать копию полей сортировочного ключа предыдущей строки состояния с противоположным значением `Sign`. Это увеличивает начальный размер хранилища, но позволяет быстро записывать данные.
2. Длинные растущие массивы в Столбцах снижают эффективность работы движка за счёт нагрузки на запись. Чем проще данные, тем выше эффективность.
3. Результаты запроса `SELECT` сильно зависят от согласованности истории изменений объекта. Будьте точны при подготовке данных для вставки. Можно получить непредсказуемые результаты для несогласованных данных, например отрицательные значения для неотрицательных метрик, таких как глубина сеанса.

## Алгоритм

Во время объединения кусков данных, каждая группа последовательных строк с одинаковым сортировочным ключом (`ORDER BY`) уменьшается до не более чем двух строк, одна из которых имеет `Sign = 1` (строка состояния), а другая строка с `Sign = -1` (строка отмены состояния). Другими словами, записи сворачиваются.

Для каждого результирующего куска данных ClickHouse сохраняет:

1. Первую строку отмены состояния и последнюю строку состояния, если количество строк обоих видов совпадает и последняя строка — строка состояния.
2. Последнюю строку состояния, если строк состояния на одну больше, чем строк отмены состояния.
3. Первую строку отмены состояния, если их на одну больше, чем строк состояния.
4. Ни одну из строк во всех остальных случаях.

Также, если строк состояния как минимум на 2 больше, чем строк отмены состояния, или, наоборот, строк отмены состояния как минимум на 2 больше, чем строк состояния, то слияние продолжается, но ClickHouse трактует подобные ситуации как логическую ошибку и записывает её в лог сервера. Подобная ошибка может возникнуть, если один и тот же блок данных вставлен несколько раз.

Как видно, от сворачивания не должны меняться результаты расчётов статистик.

Изменения постепенно сворачиваются так, что остаются лишь последнее состояние почти каждого объекта.

Столбец `Sign` необходим, поскольку алгоритм слияния не гарантирует, что все строки с одинаковым ключом сортировки будут находиться в одном результирующем куске данных и даже на одном физическом сервере. ClickHouse выполняет запросы `SELECT` несколькими потоками, и он не может предсказать порядок строк в результате. Если необходимо получить полностью свёрнутые данные из таблицы `CollapsingMergeTree`, то необходимо агрегирование.

Для завершения свертывания добавьте в запрос секцию `GROUP BY` и агрегатные функции, которые учитывают знак. Например, для расчета количества используйте `sum(Sign)` вместо `count()`. Чтобы вычислить сумму чего-либо, используйте `sum(Sign * x)` вместо `sum(x)`, и так далее, а также добавьте `HAVING sum(Sign) > 0`.

Таким образом можно вычислять агрегации `count`, `sum` и `avg`. Если объект имеет хотя бы одно не свёрнутое состояние, то может быть вычислена агрегация `uniq`. Агрегации `min` и `max` невозможно вычислить, поскольку `CollapsingMergeTree` не сохраняет историю значений свёрнутых состояний.

Если необходимо выбирать данные без агрегации (например, проверить наличие строк, последние значения которых удовлетворяют некоторым условиям), можно использовать модификатор `FINAL` для секции `FROM`. Это вариант существенно менее эффективен.

## Пример использования

Исходные данные:

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

Создание таблицы:

```
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)
```

Мы используем два запроса `INSERT` для создания двух различных кусков данных. Если вставить данные одним запросом, ClickHouse создаёт один кусок данных и никогда не будет выполнять слияние.

Получение данных:

```
SELECT * FROM UAct
```

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

UserID	PageViews	Duration	Sign
4324182021466249494	5	146	1

Что мы видим и где сворачивание?

Двумя запросами `INSERT`, мы создали два куска данных. Запрос `SELECT` был выполнен в 2 потока, и мы получили случайный порядок строк. Сворачивание не произошло, так как слияние кусков данных еще не произошло. ClickHouse объединяет куски данных в неизвестный момент времени, который мы не можем предсказать.

Таким образом, нам нужна агрегация:

```

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

Если нам не нужна агрегация, но мы хотим принудительно выполнить свёртку данных, можно использовать модификатор `FINAL` для секции `FROM`.

```
SELECT * FROM UAct FINAL
```

UserID	PageViews	Duration	Sign
4324182021466249494	6	185	1

Такой способ выбора данных очень неэффективен. Не используйте его для больших таблиц.

## Пример другого подхода

Исходные данные:

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

Идея состоит в том, что слияния при сворачивании учитывают только ключевые поля, поэтому в отменяющей строке можно указать отрицательные значения, которые нивелируют предыдущую версию записи при суммировании без учета поля `Sign`.

Для этого подхода необходимо изменить тип данных `PageViews`, `Duration` для хранения отрицательных значений `UInt8` -> `Int16`.

```

CREATE TABLE UAct
(
    UserID UInt64,
    PageViews Int16,
    Duration Int16,
    Sign Int8
)
ENGINE = CollapsingMergeTree(Sign)
ORDER BY UserID

```

Тестируем подход:

```

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; // старайтесь не использовать final (он подходит только для тестов и маленьких таблиц)

```

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

Движок:

- Позволяет быстро записывать постоянно изменяющиеся состояния объектов.
- Удаляет старые состояния объектов в фоновом режиме. Это значительно сокращает объём хранения.

Подробнее читайте в разделе [Collapsing](#).

Движок наследует функциональность от [MergeTree](#) и добавляет в алгоритм слияния кусков данных логику сворачивания (удаления) строк. [VersionedCollapsingMergeTree](#) предназначен для тех же задач, что и [CollapsingMergeTree](#), но использует другой алгоритм свёртывания, который позволяет вставлять данные в любом порядке в несколько потоков. В частности, столбец [Version](#) помогает свернуть строки правильно, даже если они вставлены в неправильном порядке. [CollapsingMergeTree](#) требует строго последовательную вставку данных.

## Создание таблицы

```
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, ...]
```

Подробности про CREATE TABLE смотрите в [описании запроса](#).

## Параметры движка

VersionedCollapsingMergeTree(sign, version)

- `sign` — Имя столбца с типом строки: `1` — строка состояния, `-1` — строка отмены состояния.

Тип данных столбца должен быть `Int8`.

- `version` — имя столбца с версией состояния объекта.

Тип данных столбца должен быть `UInt\*`.

## Секции запроса

При создании таблицы `VersionedCollapsingMergeTree` используются те же [секции](#) запроса, что и при создании таблицы `MergeTree`.

### ► Устаревший способ создания таблицы

## Сворачивание (удаление) строк

### Данные

Рассмотрим ситуацию, когда необходимо сохранять постоянно изменяющиеся данные для какого-либо объекта. Разумно иметь одну строку для объекта и обновлять эту строку при каждом изменении. Однако операция обновления является дорогостоящей и медленной для СУБД, поскольку требует перезаписи данных в хранилище. Обновление неприемлемо, если требуется быстро записывать данные, но можно записывать изменения в объект последовательно следующим образом.

Используйте столбец `Sign` при записи строки. Если `Sign = 1`, то это означает, что строка является состоянием объекта, назовём её строкой состояния. Если `Sign = -1`, то это означает отмену состояния объекта с теми же атрибутами, назовём её строкой отмены состояния. Также используйте столбец `Version`, который должен идентифицировать каждое состояние объекта отдельным номером.

Например, мы хотим рассчитать, сколько страниц пользователи посетили на каком-либо сайте и как долго они там находились. В какой-то момент времени мы записываем следующую строку состояния пользовательской активности:

UserID	PageViews	Duration	Sign	Version
4324182021466249494	5	146	1	1

Через некоторое время мы регистрируем изменение активности пользователя и записываем его следующими двумя строками.

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

Первая строка отменяет предыдущее состояние объекта (пользователя). Она должна копировать все поля отменяемого состояния за исключением `Sign`.

Вторая строка содержит текущее состояние.

Поскольку нам нужно только последнее состояние активности пользователя, строки

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

можно удалить, сворачивая (удаляя) устаревшее состояние объекта. `VersionedCollapsingMergeTree` делает это при слиянии кусков данных.

Чтобы узнать, зачем нам нужны две строки для каждого изменения, см. раздел [Алгоритм](#).

## Примечания по использованию

1. Программа, которая записывает данные, должна помнить состояние объекта, чтобы иметь возможность отменить его. Стока отмены состояния должны содержать копии полей первичного ключа и копию версии строки состояния и противоположное значение `Sign`. Это увеличивает начальный размер хранилища, но позволяет быстро записывать данные.
2. Длинные растущие массивы в столбцах снижают эффективность работы движка за счёт нагрузки на запись. Чем проще данные, тем выше эффективность.
3. `SELECT` результаты сильно зависят от согласованности истории изменений объекта. Будьте точны при подготовке данных для вставки. Вы можете получить непредсказуемые результаты с несогласованными данными, такими как отрицательные значения для неотрицательных метрик, таких как глубина сеанса.

## Алгоритм

Когда ClickHouse объединяет куски данных, он удаляет каждую пару строк, которые имеют один и тот же первичный ключ и версию и разный `Sign`. Порядок строк не имеет значения.

Когда ClickHouse вставляет данные, он упорядочивает строки по первичному ключу. Если столбец `Version` не находится в первичном ключе, ClickHouse добавляет его к первичному ключу неявно как последнее поле и использует для сортировки.

## Выборка данных

ClickHouse не гарантирует, что все строки с одинаковым первичным ключом будут находиться в одном результирующем куске данных или даже на одном физическом сервере. Это справедливо как для записи данных, так и для последующего слияния кусков данных. Кроме того, ClickHouse обрабатывает запросы SELECT несколькими потоками, и не может предсказать порядок строк в конечной выборке. Это означает, что если необходимо получить полностью «свернутые» данные из таблицы VersionedCollapsingMergeTree, то требуется агрегирование.

Для завершения свертывания добавьте в запрос секцию GROUP BY и агрегатные функции, которые учитывают знак. Например, для расчета количества используйте sum(Sign) вместо count(). Чтобы вычислить сумму чего-либо, используйте sum(Sign \* x) вместо sum(x), а также добавьте HAVING sum(Sign) > 0 .

Таким образом можно вычислять агрегации count, sum и avg. Агрегация uniq может вычисляться, если объект имеет хотя бы одно не свернутое состояние. Невозможно вычислить агрегации min и max поскольку VersionedCollapsingMergeTree не сохраняет историю значений для свернутых состояний.

Если необходимо выбирать данные без агрегации (например, проверить наличие строк, последние значения которых удовлетворяют некоторым условиям), можно использовать модификатор FINAL для секции FROM. Такой подход неэффективен и не должен использоваться с большими таблицами.

## Пример использования

Данные для примера:

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

Создание таблицы:

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

Вставка данных:

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

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

Мы используем два запроса INSERT для создания двух различных кусков данных. Если мы вставляем данные с помощью одного запроса, ClickHouse создаёт один кусок данных и не будет выполнять слияние.

Получение данных:

```
SELECT * FROM UAct
```

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

Что мы видим и где сворачивание?

Мы создали два куска данных, используя два запроса `INSERT`. Запрос `SELECT` был выполнен в два потока, и результатом является случайный порядок строк.

Свертывание не произошло, поскольку части данных еще не были объединены. ClickHouse объединяет части данных в неизвестный момент времени, который мы не можем предсказать.

Поэтому нам нужна агрегация:

```
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

Если нам не нужна агрегация, но мы хотим принудительно выполнить свёртку данных, то можно использовать модификатор `FINAL` для секции `FROM`.

```
SELECT * FROM UAct FINAL
```

UserID	PageViews	Duration	Sign	Version
4324182021466249494	6	185	1	2

Это очень неэффективный способ выбора данных. Не используйте его для больших таблиц.

## GraphiteMergeTree

Движок предназначен для прореживания и агрегирования/усреднения (`rollup`) данных `Graphite`. Он может быть интересен разработчикам, которые хотят использовать ClickHouse как хранилище данных для `Graphite`.

Если `rollup` не требуется, то для хранения данных `Graphite` можно использовать любой движок таблиц ClickHouse, в противном случае используйте `GraphiteMergeTree`. Движок уменьшает объём хранения и повышает эффективность запросов от `Graphite`.

Движок наследует свойства от `MergeTree`.

## Создание таблицы

```
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, ...]
```

Смотрите описание запроса [CREATE TABLE](#).

В таблице должны быть столбцы для следующих данных:

- Название метрики (сенсора Graphite). Тип данных: `String`.
- Время измерения метрики. Тип данных `DateTime`.
- Значение метрики. Тип данных: любой числовой.
- Версия метрики. Тип данных: любой числовой (ClickHouse сохраняет строки с последней версией или последнюю записанную строку, если версии совпадают. Другие строки удаляются при слиянии кусков данных).

Имена этих столбцов должны быть заданы в конфигурации `rollup`.

### Параметры `GraphiteMergeTree`

- `config_section` — имя раздела в конфигурационном файле, в котором находятся правила `rollup`.

### Секции запроса

При создании таблицы `GraphiteMergeTree` используются те же [секции](#) запроса, что и при создании таблицы `MergeTree`.

#### ► Устаревший способ создания таблицы

## Конфигурация Rollup

Настройки прореживания данных задаются параметром `graphite_rollup` в конфигурации сервера . Имя параметра может быть любым. Можно создать несколько конфигураций и использовать их для разных таблиц.

Структура конфигурации `rollup`:

```
required-columns
patterns
```

### Требуемые столбцы (`required-columns`)

- `path_column_name` — столбец, в котором хранится название метрики (сенсор Graphite). Значение по умолчанию: `Path`.
- `time_column_name` — столбец, в котором хранится время измерения метрики. Значение по умолчанию: `Time`.

- `value_column_name` — столбец со значением метрики в момент времени, установленный в `time_column_name`. Значение по умолчанию: `Value`.
- `version_column_name` — столбец, в котором хранится версия метрики. Значение по умолчанию: `Timestamp`.

## Правила (patterns)

Структура раздела `patterns`:

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

## Внимание

Правила должны быть строго упорядочены:

1. Правила **без** `function` **или** `retention`.
2. Правила одновременно содержащие `function` **и** `retention`.
3. Правило `default`.

При обработке строки ClickHouse проверяет правила в разделе `pattern`. Каждый `pattern` (включая `default`) может содержать параметр агрегации `function`, параметр `retention`, или оба параметра одновременно. Если имя метрики соответствует шаблону `regexp`, то применяются правила `pattern`, в противном случае правило `default`.

Поля для разделов `pattern` и `default`:

- `regexp` – шаблон имени метрики.
- `age` – минимальный возраст данных в секундах.
- `precision` – точность определения возраста данных в секундах. Должен быть делителем для 86400 (количество секунд в сутках).
- `function` – имя агрегирующей функции, которую следует применить к данным, чей возраст оказался в интервале `[age, age + precision]`. Допустимые функции: `min/max/any/avg`. `Avg` вычисляется неточно, как среднее от средних.

## Пример конфигурации

```
<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>
```

## Внимание

Прореживание данных производится во время слияний. Обычно для старых партиций слияния не запускаются, поэтому для прореживания надо инициировать незапланированное слияние используя **optimize**. Или использовать дополнительные инструменты, например **graphite-ch-optimizer**.

## Семейство Log

Движки разработаны для сценариев, когда необходимо быстро записывать много таблиц с небольшим объёмом данных (менее 1 миллиона строк), а затем читать их целиком.

Движки семейства:

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

Табличные движки семейства Log могут хранить данные в распределенных файловых системах **HDFS** или **S3**.

## Общие свойства

Движки:

- Хранят данные на диске.
- Добавляют данные в конец файла при записи.

- Поддерживают блокировки для конкурентного доступа к данным.

Во время запросов `INSERT` таблица блокируется, а другие запросы на чтение и запись ожидают разблокировки таблицы. Если запросов на запись данных нет, то можно выполнять любое количество конкретных запросов на чтение.

- Не поддерживают операции [мутации](#).
- Не поддерживают индексы.

Это означает, что запросы `SELECT` не эффективны для выборки диапазонов данных.

- Записывают данные не атомарно.

Вы можете получить таблицу с повреждёнными данными, если что-то прервёт операцию записи (например, аварийное завершение работы сервера).

## Отличия

Двигок `TinyLog` самый простой в семье и обеспечивает самые низкие функциональность и эффективность. Двигок `TinyLog` не поддерживает параллельного чтения данных в несколько потоков. Двигок читает данные медленнее, чем оба других движка с параллельным чтением, и использует почти столько же дескрипторов, сколько и движок `Log`, поскольку хранит каждый столбец в отдельном файле. Его можно использовать в простых сценариях с низкой нагрузкой.

Движки `Log` и `StripeLog` поддерживают параллельное чтение. При чтении данных, ClickHouse использует множество потоков. Каждый поток обрабатывает отдельный блок данных. Двигок `Log` сохраняет каждый столбец таблицы в отдельном файле. Двигок `StripeLog` хранит все данные в одном файле. Таким образом, движок `StripeLog` использует меньше дескрипторов в операционной системе, а движок `Log` обеспечивает более эффективное считывание данных.

## StripeLog

Двигок относится к семейству движков `Log`. Смотрите общие свойства и различия движков в статье [Семейство Log](#).

Двигок разработан для сценариев, когда необходимо записывать много таблиц с небольшим объёмом данных (менее 1 миллиона строк).

## Создание таблицы

```
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
```

Смотрите подробное описание запроса [CREATE TABLE](#).

## Запись данных

Двигок `StripeLog` хранит все столбцы в одном файле. При каждом запросе `INSERT`, ClickHouse добавляет блок данных в конец файла таблицы, записывая столбцы один за другим.

Для каждой таблицы ClickHouse записывает файлы:

- `data.bin` — файл с данными.
- `index.mrk` — файл с метками. Метки содержат смещения для каждого столбца каждого вставленного блока данных.

Двигок `StripeLog` не поддерживает запросы `ALTER UPDATE` и `ALTER DELETE`.

## Чтение данных

Файл с метками позволяет ClickHouse распараллеливать чтение данных. Это означает, что запрос `SELECT` возвращает строки в непредсказуемом порядке. Используйте секцию `ORDER BY` для сортировки строк.

## Пример использования

Создание таблицы:

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

Вставка данных:

```
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')
```

Мы использовали два запроса `INSERT` для создания двух блоков данных внутри файла `data.bin`.

ClickHouse использует несколько потоков при выборе данных. Каждый поток считывает отдельный блок данных и возвращает результирующие строки независимо по мере завершения. В результате порядок блоков строк в выходных данных в большинстве случаев не совпадает с порядком тех же блоков во входных данных. Например:

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

Сортировка результатов (по умолчанию по возрастанию):

```
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

Движок относится к семейству движков Log. Смотрите общие свойства и различия движков в статье [Семейство Log](#).

Отличается от [TinyLog](#) тем, что вместе с файлами столбцов лежит небольшой файл "засечек". Засечки пишутся на каждый блок данных и содержат смещение: с какого места нужно читать файл, чтобы пропустить заданное количество строк. Это позволяет читать данные из таблицы в несколько потоков.

При конкурентном доступе к данным чтения могут выполняться одновременно, а записи блокируют чтения и друг друга.

Движок Log не поддерживает индексы. Также, если при записи в таблицу произошёл сбой, то таблица станет битой, и чтения из нее будут возвращать ошибку. Движок Log подходит для временных данных, write-once таблиц, а также для тестовых и демонстрационных целей.

# TinyLog

Движок относится к семейству движков Log. Смотрите общие свойства и различия движков в статье [Семейство Log](#).

Типичный способ использования этой движка — это write-once: сначала данные один раз записываются, а затем читаются столько раз, сколько это необходимо. Например, можно использовать таблицы с движком TinyLog для хранения промежуточных данных, которые обрабатываются небольшими блоками. Учтите, что хранить данные в большом количестве мелких таблиц неэффективно.

Запросы выполняются в один поток. То есть, этот движок предназначен для сравнительно маленьких таблиц (до 1 000 000 строк). Этот движок таблиц имеет смысл использовать в том случае, когда у вас есть много маленьких таблиц, так как он проще, чем движок Log (требуется открывать меньше файлов).

# ODBC

Позволяет ClickHouse подключаться к внешним базам данных с помощью [ODBC](#).

Чтобы использование ODBC было безопасным, ClickHouse использует отдельную программу `clickhouse-odbc-bridge`. Если драйвер ODBC подгружать непосредственно из `clickhouse-server`, то проблемы с драйвером могут привести к аварийной остановке сервера ClickHouse. ClickHouse автоматически запускает `clickhouse-odbc-bridge` по мере необходимости. Программа устанавливается из того же пакета, что и `clickhouse-server`.

Движок поддерживает тип данных [Nullable](#).

## Создание таблицы

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

Смотрите подробное описание запроса [CREATE TABLE](#).

Структура таблицы может отличаться от структуры исходной таблицы в удалённой СУБД:

- Имена столбцов должны быть такими же, как в исходной таблице, но вы можете использовать только некоторые из этих столбцов и в любом порядке.
- Типы столбцов могут отличаться от типов аналогичных столбцов в исходной таблице. ClickHouse пытается [приводить](#) значения к типам данных ClickHouse.
- Настройка `external_table_functions_use_nulls` определяет как обрабатывать Nullable столбцы. Значение по умолчанию: 1. Если значение 0, то табличная функция не делает Nullable столбцы, а вместо NULL выставляет значения по умолчанию для скалярного типа. Это также применимо для значений NULL внутри массивов.

## Параметры движка

- `connection_settings` — название секции с настройками соединения в файле `odbc.ini`.
- `external_database` — имя базы данных во внешней СУБД.
- `external_table` — имя таблицы в `external_database`.

## Пример использования

### Извлечение данных из локальной установки MySQL через ODBC

Этот пример проверялся в Ubuntu Linux 18.04 для MySQL server 5.7.

Убедитесь, что unixODBC и MySQL Connector установлены.

По умолчанию (если установлен из пакетов) ClickHouse запускается от имени пользователя `clickhouse`. Таким образом, вам нужно создать и настроить этого пользователя на сервере MySQL.

```
$ sudo mysql
```

```
mysql> CREATE USER 'clickhouse'@'localhost' IDENTIFIED BY 'clickhouse';
mysql> GRANT ALL PRIVILEGES ON *.* TO 'clickhouse'@'clickhouse' WITH GRANT OPTION;
```

Теперь настроим соединение в `/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
```

Вы можете проверить соединение с помощью утилиты `isql` из установки unixODBC.

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

Таблица в 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)
```

Таблица в ClickHouse, которая получает данные из таблицы MySQL:

```
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

## Смотрите также

- [Внешние словари ODBC](#)
- [Табличная функция odbc](#)

## JDBC

Позволяет ClickHouse подключаться к внешним базам данных с помощью [JDBC](#).

Для реализации соединения по JDBC ClickHouse использует отдельную программу [clickhouse-jdbc-bridge](#), которая должна запускаться как демон.

Движок поддерживает тип данных [Nullable](#).

## Создание таблицы

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name
ENGINE = JDBC(datasource_uri, external_database, external_table)
```

## Параметры движка

- `datasource_uri` — URI или имя внешней СУБД.

URI Формат: `jdbc:<driver_name>://<host_name>:<port>/?user=<username>&password=<password>`.

Пример для MySQL: `jdbc:mysql://localhost:3306/?user=root&password=root`.

- `external_database` — база данных во внешней СУБД.
- `external_table` — таблицы в `external_database` или запросе выбора, например `select * from table1, где column1 = 1`.

## Пример использования

Создадим таблицу в на сервере MySQL с помощью консольного клиента 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)
```

Создадим таблицу на сервере ClickHouse и получим из неё данные:

```
CREATE TABLE jdbc_table ENGINE JDBC('jdbc:mysql://localhost:3306/?user=root&password=root', 'test', 'test')
```

```
DESCRIBE TABLE jdbc_table
```

name	type	default_type	default_expression
int_id	Int32		
int_nullable	Nullable(Int32)		
float	Float32		
float_nullable	Nullable(Float32)		

```
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
```

## Смотрите также

- Табличная функция JDBC.

# MySQL

Движок MySQL позволяет выполнять запросы SELECT и INSERT над данными, хранящимися на удалённом MySQL сервере.

## Создание таблицы

```
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']);
```

Смотрите подробное описание запроса [CREATE TABLE](#).

Структура таблицы может отличаться от структуры исходной таблицы MySQL:

- Имена столбцов должны быть такими же, как в исходной таблице MySQL, но можно использовать только некоторые из этих столбцов и в любом порядке.
- Типы столбцов могут отличаться от типов в исходной таблице MySQL. ClickHouse пытается [привести](#) значения к типам данных ClickHouse.
- Настройка [external\\_table\\_functions\\_use\\_nulls](#) определяет как обрабатывать Nullable столбцы. Значение по умолчанию: 1. Если значение 0, то табличная функция не делает Nullable столбцы, а вместо NULL выставляет значения по умолчанию для скалярного типа. Это также применимо для значений NULL внутри массивов.

## Параметры движка

- `host:port` — адрес сервера MySQL.
- `database` — имя базы данных на удалённом сервере.
- `table` — имя таблицы на удалённом сервере.
- `user` — пользователь MySQL.
- `password` — пароль пользователя.
- `replace_query` — флаг, отвечающий за преобразование запросов `INSERT INTO` в `REPLACE INTO`. Если `replace_query=1`, то запрос заменяется.
- `on_duplicate_clause` — выражение `ON DUPLICATE KEY on_duplicate_clause`, добавляемое к запросу `INSERT`.

Пример: `INSERT INTO t (c1,c2) VALUES ('a', 2) ON DUPLICATE KEY UPDATE c2 = c2 + 1`, где `on\_duplicate\_clause` это `UPDATE c2 = c2 + 1`. Чтобы узнать какие `on\_duplicate\_clause` можно использовать с секцией `ON DUPLICATE KEY` обратитесь к [документации MySQL](<https://dev.mysql.com/doc/refman/8.0/en/insert-on-duplicate.html>).

Чтобы указать `on\_duplicate\_clause` необходимо передать `0` в параметр `replace\_query`. Если одновременно передать `replace\_query = 1` и `on\_duplicate\_clause`, то ClickHouse сгенерирует исключение.

Простые условия WHERE такие как `=`, `!=`, `>`, `>=`, `<`, `=` выполняются на стороне сервера MySQL.

Остальные условия и ограничение выборки `LIMIT` будут выполнены в ClickHouse только после выполнения запроса к MySQL.

Поддерживает несколько реплик, которые должны быть перечислены через `|`. Например:

```
CREATE TABLE test_replicas (id UInt32, name String, age UInt32, money UInt32) ENGINE = MySQL('mysql{2|3|4}:3306', 'clickhouse', 'test_replicas', 'root', 'clickhouse');
```

## Пример использования

Таблица в 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 |
+-----+-----+-----+
|     1 |      NULL |    2 |
+-----+-----+-----+
1 row in set (0,00 sec)
```

Таблица в ClickHouse, которая получает данные из созданной ранее таблицы MySQL:

```
CREATE TABLE mysql_table
(
  `float_nullable` Nullable(Float32),
  `int_id` Int32
)
ENGINE = MySQL('localhost:3306', 'test', 'test', 'bayonet', '123')
```

```
SELECT * FROM mysql_table
```

float_nullable	int_id
NULL	1

## Смотрите также

- [Табличная функция 'mysql'](#)
- [Использование MySQL в качестве источника для внешнего словаря](#)

## Движок таблиц S3

Этот движок обеспечивает интеграцию с экосистемой [Amazon S3](#). Он похож на движок [HDFS](#), но обеспечивает специфические для S3 возможности.

## Создание таблицы

```
CREATE TABLE s3_engine_table (name String, value UInt32)
ENGINE = S3(path, [aws_access_key_id, aws_secret_access_key], format, [compression])
```

## Параметры движка

- `path` — URL-адрес бакета с указанием пути к файлу. Поддерживает следующие подстановочные знаки в режиме "только чтение": `*`, `?`, `{abc,def}` и `{N..M}` где `N, M` — числа, `'abc'`, `'def'` — строки.  
Подробнее смотри [ниже](#).
- `format` — [формат](#) файла.
- `aws_access_key_id, aws_secret_access_key` - данные пользователя учетной записи [AWS](#). Вы можете использовать их для аутентификации ваших запросов. Необязательный параметр. Если параметры учетной записи не указаны, то используются данные из конфигурационного файла. Смотрите подробнее [Использование сервиса S3 для хранения данных](#).
- `compression` — тип сжатия. Возможные значения: `none, gzip/gz, brotli/br, xz/LZMA, zstd/zst`. Необязательный параметр. Если не указано, то тип сжатия определяется автоматически по расширению файла.

## Пример

```
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');
INSERT INTO s3_engine_table VALUES ('one', 1), ('two', 2), ('three', 3);
SELECT * FROM s3_engine_table LIMIT 2;
```

name	value
one	1
two	2

## Виртуальные столбцы

- `_path` — путь к файлу.
- `_file` — имя файла.

Подробнее про виртуальные столбцы можно прочитать [здесь](#).

## Детали реализации

- Чтение и запись могут быть параллельными.
- Поддерживается репликация без копирования данных ([zero-copy](#)).
- Не поддерживаются:
  - запросы `ALTER` и `SELECT...SAMPLE`,
  - индексы.

## Символы подстановки

Аргумент `path` может указывать на несколько файлов, используя подстановочные знаки. Для обработки файл должен существовать и соответствовать всему шаблону пути. Список файлов определяется во время выполнения запроса `SELECT` (не в момент выполнения запроса `CREATE`).

- `*` — заменяет любое количество любых символов, кроме `/`, включая пустую строку.

- `? —` заменяет любые одиночные символы.
- `{some_string, another_string, yet_another_one}` — заменяет любые строки `'some_string', 'another_string', 'yet_another_one'`.
- `{N..M}` — заменяет любое число от N до M, включая обе границы. N и M могут иметь ведущие нули, например `000..078`.

Конструкции с `{}` аналогичны функции `remote`.

## Настройки движка S3

Перед выполнением запроса или в конфигурационном файле могут быть установлены следующие настройки:

- `s3_max_single_part_upload_size` — максимальный размер объекта для загрузки с использованием однокомпонентной загрузки в S3. Значение по умолчанию — 64 Мб.
- `s3_min_upload_part_size` — минимальный размер объекта для загрузки при многокомпонентной загрузке в [S3 Multipart upload](#). Значение по умолчанию — 512 Мб.
- `s3_max_redirects` — максимальное количество разрешенных переадресаций S3. Значение по умолчанию — 10.
- `s3_single_read_retries` — максимальное количество попыток запроса при единичном чтении. Значение по умолчанию — 4.

Соображение безопасности: если злонамеренный пользователь попробует указать произвольные URL-адреса S3, параметр `s3_max_redirects` должен быть установлен в ноль, чтобы избежать атак [SSRF](#). Как альтернатива, в конфигурации сервера должен быть указан `remote_host_filter`.

## Настройки точки приема запроса

Для точки приема запроса (которая соответствует точному префиксу URL-адреса) в конфигурационном файле могут быть заданы следующие настройки:

Обязательная настройка:

- `endpoint` — указывает префикс точки приема запроса.

Необязательные настройки:

- `access_key_id` и `secret_access_key` — указывают учетные данные для использования с данной точкой приема запроса.
- `use_environment_credentials` — если `true`, S3-клиент будет пытаться получить учетные данные из переменных среды и метаданных [Amazon EC2](#) для данной точки приема запроса. Значение по умолчанию — `false`.
- `use_insecure_imds_request` — признак использования менее безопасного соединения при выполнении запроса к IMDS при получении учётных данных из метаданных Amazon EC2. Значение по умолчанию — `false`.
- `region` — название региона S3.
- `header` — добавляет указанный HTTP-заголовок к запросу на заданную точку приема запроса. Может быть определен несколько раз.
- `server_side_encryption_customer_key_base64` — устанавливает необходимые заголовки для доступа к объектам S3 с шифрованием SSE-C.
- `single_read_retries` — Максимальное количество попыток запроса при единичном чтении. Значение по умолчанию — 4.

## Пример

```
<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> -->
    <!-- <single_read_retries>4</single_read_retries> -->
  </endpoint-name>
</s3>
```

## Примеры использования

Предположим, у нас есть несколько файлов в формате CSV со следующими URL-адресами в 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. Существует несколько способов создать таблицу, включающую в себя все шесть файлов:

```
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. Другой способ:

```
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. Таблица содержит все файлы в обоих каталогах (все файлы должны соответствовать формату и схеме, описанным в запросе):

```
CREATE TABLE table_with_asterisk (name String, value UInt32)
ENGINE = S3('https://storage.yandexcloud.net/my-test-bucket-768/{some,another}_prefix/*', 'CSV');
```

Если список файлов содержит диапазоны чисел с ведущими нулями, используйте конструкцию с фигурными скобками для каждой цифры отдельно или используйте ?.

4. Создание таблицы из файлов с именами 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');
```

## Смотрите также

- [Табличная функция s3](#)

# MongoDB

Движок таблиц MongoDB позволяет читать данные из коллекций СУБД MongoDB. В таблицах допустимы только плоские (не вложенные) типы данных. Запись (`INSERT`-запросы) не поддерживается.

## Создание таблицы

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name
(
    name1 [type1],
    name2 [type2],
    ...
) ENGINE = MongoDB(host:port, database, collection, user, password [, options]);
```

### Параметры движка

- `host:port` — адрес сервера MongoDB.
- `database` — имя базы данных на удалённом сервере.
- `collection` — имя коллекции на удалённом сервере.
- `user` — пользователь MongoDB.
- `password` — пароль пользователя.
- `options` — MongoDB connection string options (optional parameter).

## Примеры использования

Создание таблицы в ClickHouse для чтения данных из коллекции MongoDB:

```
CREATE TABLE mongo_table
(
    key UInt64,
    data String
) ENGINE = MongoDB('mongo1:27017', 'test', 'simple_table', 'testuser', 'clickhouse');
```

Чтение с сервера MongoDB, защищенного SSL:

```
CREATE TABLE mongo_table_ssl
(
    key UInt64,
    data String
) ENGINE = MongoDB('mongo2:27017', 'test', 'simple_table', 'testuser', 'clickhouse', 'ssl=true');
```

Запрос к таблице:

```
SELECT COUNT() FROM mongo_table;
```

```
count()
4 |
```

# HDFS

Управляет данными в HDFS. Данный движок похож на движки [File](#) и [URL](#).

## Использование движка

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

В параметр `URI` нужно передавать полный URI файла в HDFS.

Параметр `format` должен быть таким, который ClickHouse может использовать и в запросах `INSERT`, и в запросах `SELECT`. Полный список поддерживаемых форматов смотрите в разделе [Форматы](#).

Часть URI с путем файла может содержать шаблоны. В этом случае таблица может использоваться только для чтения.

### Пример:

1. Создадим на сервере таблицу `hdfs_engine_table`:

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

2. Заполним файл:

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

3. Запросим данные:

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

name	value
one	1
two	2

## Детали реализации

- Поддерживается многопоточное чтение и запись.
- Поддерживается репликация без копирования данных ([zero-copy](#)).
- Не поддерживается:
  - использование операций `ALTER` и `SELECT...SAMPLE`;
  - индексы.

### Шаблоны в пути

Шаблоны могут содержаться в нескольких компонентах пути. Обрабатываются только существующие файлы, название которых целиком удовлетворяет шаблону (не только суффиксом или префиксом).

- `*` — Заменяет любое количество любых символов кроме `/`, включая отсутствие символов.
- `?` — Заменяет ровно один любой символ.
- `{some_string,another_string,yet_another_one}` — Заменяет любую из строк `'some_string'`, `'another_string'`, `'yet_another_one'`.

- {N..M} — Заменяет любое число в интервале от N до M включительно (может содержать ведущие нули).

Конструкция с {} аналогична табличной функции **remote**.

## Пример

1. Предположим, у нас есть несколько файлов со следующими URI в 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. Есть несколько возможностей создать таблицу, состоящую из этих шести файлов:

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

Другой способ:

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

Таблица, состоящая из всех файлов в обеих директориях (все файлы должны удовлетворять формату и схеме, указанной в запросе):

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

## Warning

Если список файлов содержит числовые интервалы с ведущими нулями, используйте конструкцию с фигурными скобочками для каждой цифры или используйте ?.

## Example

Создадим таблицу с именами 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')
```

## Конфигурация

Похоже на GraphiteMergeTree, движок HDFS поддерживает расширенную конфигурацию с использованием файла конфигурации ClickHouse. Есть два раздела конфигурации которые вы можете использовать: глобальный (`hdfs`) и на уровне пользователя (`hdfs_*`). Глобальные настройки применяются первыми, и затем применяется конфигурация уровня пользователя (если она указана).

```
<!-- Глобальные настройки для движка HDFS -->
<dfs>
  <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>
</dfs>

<!-- Конфигурация специфичная для пользователя "root" -->
<dfs_root>
  <hadoop_kerberos_principal>root@TEST.CLICKHOUSE.TECH</hadoop_kerberos_principal>
</dfs_root>
```

## Параметры конфигурации

Поддерживаемые из libhdfs3

| **параметр | по умолчанию |**

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 | "" |
```

Руководство по конфигурации [HDFS](#) поможет объяснить назначения некоторых параметров.

## Расширенные параметры для ClickHouse

```
| параметр | по умолчанию |
|hadoop_kerberos_keytab | "" |
|hadoop_kerberos_principal | "" |
|hadoop_kerberos_kinit_command | kinit |
```

## Ограничения

- `hadoop_security_kerberos_ticket_cache_path` могут быть определены только на глобальном уровне

## Поддержка Kerberos

Если `hadoop_security_authentication` параметр имеет значение 'kerberos', ClickHouse аутентифицируется с помощью Kerberos.

[Расширенные параметры](#) и `hadoop_security_kerberos_ticket_cache_path` помогают сделать это.

Обратите внимание что из-за ограничений libhdfs3 поддерживается только устаревший метод аутентификации,

коммуникация с узлами данных не защищена SASL (HADOOP\_SECURE\_DN\_USER надежный показатель такого подхода к безопасности). Используйте `tests/integration/test_storage_kerberized_hdfs/hdfs_configs/bootstrap.sh` для примера настроек.

Если `hadoop_kerberos_keytab`, `hadoop_kerberos_principal` или `hadoop_kerberos_kinit_command` указаны в настройках, `kinit` будет вызван. `hadoop_kerberos_keytab` и `hadoop_kerberos_principal` обязательны в этом случае. Необходимо также будет установить `kinit` и файлы конфигурации krb5.

## Виртуальные столбцы

- `_path` — Путь к файлу.
- `_file` — Имя файла.

### См. также

- [Виртуальные колонки](#)

## SQLite

Движок позволяет импортировать и экспортить данные из SQLite, а также поддерживает отправку запросов к таблицам SQLite напрямую из ClickHouse.

## Создание таблицы

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name
(
    name1 [type1],
    name2 [type2], ...
) ENGINE = SQLite('db_path', 'table')
```

### Параметры движка

- `db_path` — путь к файлу с базой данных SQLite.
- `table` — имя таблицы в базе данных SQLite.

## Примеры использования

Отобразим запрос, с помощью которого была создана таблица SQLite:

```
SHOW CREATE TABLE sqlite_db.table2;
```

```
CREATE TABLE SQLite.table2
(
    `col1` Nullable(Int32),
    `col2` Nullable(String)
)
ENGINE = SQLite('sqlite.db','table2');
```

Получим данные из таблицы:

```
SELECT * FROM sqlite_db.table2 ORDER BY col1;
```

col1	col2
1	text1
2	text2
3	text3

### См. также

- [SQLite](#) движок баз данных
- [sqlite](#) табличная функция

## Kafka

Движок работает с [Apache Kafka](#).

Kafka позволяет:

- Публиковать/подписываться на потоки данных.
- Организовать отказоустойчивое хранилище.
- Обрабатывать потоки по мере их появления.

## Создание таблицы

```

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_skip_broken_messages = N]
    [kafka_commit_every_batch = 0,]
    [kafka_thread_per_consumer = 0]

```

Обязательные параметры:

- `kafka_broker_list` — перечень брокеров, разделенный запятыми (`localhost:9092`).
- `kafka_topic_list` — перечень необходимых топиков Kafka.
- `kafka_group_name` — группа потребителя Kafka. Отступы для чтения отслеживаются для каждой группы отдельно. Если необходимо, чтобы сообщения не повторялись на кластере, используйте **всезде одно имя группы**.
- `kafka_format` — формат сообщений. Названия форматов должны быть теми же, что можно использовать в секции FORMAT, например, `JSONEachRow`. Подробнее читайте в разделе [Форматы](#).

Опциональные параметры:

- `kafka_row_delimiter` — символ-разделитель записей (строк), которым завершается сообщение.
- `kafka_schema` — опциональный параметр, необходимый, если используется формат, требующий определения схемы. Например, `Cap'n Proto` требует путь к файлу со схемой и название корневого объекта `schema.capnp:Message`.
- `kafka_num_consumers` — количество потребителей (consumer) на таблицу. По умолчанию: `1`. Укажите больше потребителей, если пропускная способность одного потребителя недостаточна. Общее число потребителей не должно превышать количество партиций в топике, так как на одну партицию может быть назначено не более одного потребителя.
- `kafka_max_block_size` — максимальный размер пачек (в сообщениях) для `roll` (по умолчанию `max_block_size`).
- `kafka_skip_broken_messages` — максимальное количество некорректных сообщений в блоке. Если `kafka_skip_broken_messages = N`, то движок отбрасывает `N` сообщений Кафки, которые не получилось обработать. Одно сообщение в точности соответствует одной записи (строке). Значение по умолчанию – `0`.
- `kafka_commit_every_batch` — включает или отключает режим записи каждой принятой и обработанной пачки по отдельности вместо единой записи целого блока (по умолчанию `0`).
- `kafka_thread_per_consumer` — включает или отключает предоставление отдельного потока каждому потребителю (по умолчанию `0`). При включенном режиме каждый потребитель сбрасывает данные независимо и параллельно, при отключённом — строки с данными от нескольких потребителей собираются в один блок.

Примеры

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

SELECT * FROM queue LIMIT 5;

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

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

## ► Устаревший способ создания таблицы

## Описание

Полученные сообщения отслеживаются автоматически, поэтому из одной группы каждое сообщение считывается только один раз. Если необходимо получить данные дважды, то создайте копию таблицы с другим именем группы.

Группы пластичны и синхронизированы на кластере. Например, если есть 10 топиков и 5 копий таблицы в кластере, то в каждую копию попадет по 2 топика. Если количество копий изменится, то распределение топиков по копиям изменится автоматически. Подробно читайте об этом на <http://kafka.apache.org/intro>.

Чтение сообщения с помощью `SELECT` не слишком полезно (разве что для отладки), поскольку каждое сообщения может быть прочитано только один раз. Практичнее создавать потоки реального времени с помощью материализованных представлений. Для этого:

1. Создайте потребителя Kafka с помощью движка и рассматривайте его как поток данных.
2. Создайте таблицу с необходимой структурой.
3. Создайте материализованное представление, которое преобразует данные от движка и помещает их в ранее созданную таблицу.

Когда к движку присоединяется материализованное представление (`MATERIALIZED VIEW`), оно начинает в фоновом режиме собирать данные. Это позволяет непрерывно получать сообщения от Kafka и преобразовывать их в необходимый формат с помощью `SELECT`.

Материализованных представлений у одной kafka таблицы может быть сколько угодно, они не считывают данные из таблицы kafka непосредственно, а получают новые записи (блоками), таким образом можно писать в несколько таблиц с разным уровнем детализации (с группировкой - агрегацией и без).

Пример:

```

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;

```

Для улучшения производительности полученные сообщения группируются в блоки размера `max_insert_block_size`. Если блок не удалось сформировать за `stream_flush_interval_ms` миллисекунд, то данные будут сброшены в таблицу независимо от полноты блока.

Чтобы остановить получение данных топика или изменить логику преобразования, отсоедините материализованное представление:

```

DETACH TABLE consumer;
ATTACH TABLE consumer;

```

Если необходимо изменить целевую таблицу с помощью `ALTER`, то материализованное представление рекомендуется отключить, чтобы избежать несостыковки между целевой таблицей и данными от представления.

## Конфигурация

Аналогично GraphiteMergeTree, движок Kafka поддерживает расширенную конфигурацию с помощью конфигурационного файла ClickHouse. Существует два конфигурационных ключа, которые можно использовать: глобальный (`kafka`) и по топикам (`kafka_topic_*`). Сначала применяется глобальная конфигурация, затем конфигурация по топикам (если она существует).

```

<!-- 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>

```

В документе [librdkafka configuration reference](#) можно увидеть список возможных опций конфигурации. Используйте подчеркивание (`_`) вместо точки в конфигурации ClickHouse. Например, `checkcrcs=true` будет соответствовать `<check_crcs>true</check_crcs>`.

## Поддержка Kerberos

Чтобы начать работу с Kafka с поддержкой Kerberos, добавьте дочерний элемент `security_protocol` со значением `sasl_plaintext`. Этого будет достаточно, если получен тикет на получение тикета (ticket-granting ticket) Kerberos и он кэшируется средствами ОС.

ClickHouse может поддерживать учетные данные Kerberos с помощью файла keytab. Рассмотрим дочерние элементы `sasl_kerberos_service_name`, `sasl_kerberos_keytab`, `sasl_kerberos_principal` и `sasl.kerberos.kinit.cmd`.

Пример:

```
<!-- 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>
```

## Виртуальные столбцы

- `_topic` — топик Kafka.
- `_key` — ключ сообщения.
- `_offset` — оффсет сообщения.
- `_timestamp` — временная метка сообщения.
- `_partition` — секция топика Kafka.

### Смотрите также

- [Виртуальные столбцы](#)
- [background\\_schedule\\_pool\\_size](#)

## Двигок EmbeddedRocksDB

Этот движок позволяет интегрировать ClickHouse с [rocksdb](#).

## Создание таблицы

```
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);
```

Обязательные параметры:

- `primary_key_name` может быть любое имя столбца из списка столбцов.
- Указание первичного ключа `primary key` является обязательным. Он будет сериализован в двоичном формате как ключ `rocksdb`.
- Поддерживается только один столбец в первичном ключе.
- Столбцы, которые отличаются от первичного ключа, будут сериализованы в двоичном формате как значение `rockdb` в соответствующем порядке.

- Запросы с фильтрацией по ключу `equals` или `in` оптимизируются для поиска по нескольким ключам из `rocksdb`.

Пример:

```
CREATE TABLE test
(
    `key` String,
    `v1` UInt32,
    `v2` String,
    `v3` Float32,
)
ENGINE = EmbeddedRocksDB
PRIMARY KEY key;
```

## PostgreSQL

Двигок PostgreSQL позволяет выполнять запросы `SELECT` и `INSERT` для таблиц на удаленном сервере PostgreSQL.

### Создание таблицы

```
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`]);
```

Смотрите подробное описание запроса [CREATE TABLE](#).

Структура таблицы может отличаться от структуры исходной таблицы PostgreSQL:

- Имена столбцов должны быть такими же, как в исходной таблице PostgreSQL, но можно использовать только некоторые из этих столбцов и в любом порядке.
- Типы столбцов могут отличаться от типов в исходной таблице PostgreSQL. ClickHouse пытается [привести](#) значения к типам данных ClickHouse.
- Настройка `external_table_functions_use_nulls` определяет как обрабатывать Nullable столбцы. Значение по умолчанию: 1. Если значение 0, то табличная функция не делает Nullable столбцы, а вместо NULL выставляет значения по умолчанию для скалярного типа. Это также применимо для значений NULL внутри массивов.

### Параметры движка

- `host:port` — адрес сервера PostgreSQL.
- `database` — имя базы данных на сервере PostgreSQL.
- `table` — имя таблицы.
- `user` — имя пользователя PostgreSQL.
- `password` — пароль пользователя PostgreSQL.
- `schema` — имя схемы, если не используется схема по умолчанию. Необязательный аргумент.

### Особенности реализации

Запросы `SELECT` на стороне PostgreSQL выполняются как `COPY (SELECT ...)` TO STDOUT внутри транзакции PostgreSQL только на чтение с коммитом после каждого запроса `SELECT`.

Простые условия для `WHERE`, такие как `=`, `!=`, `>`, `>=`, `<`, `<=` и `IN`, исполняются на стороне PostgreSQL сервера.

Все операции объединения, агрегации, сортировки, условия `IN [ array ]` и ограничения `LIMIT` выполняются на стороне ClickHouse только после того, как запрос к PostgreSQL закончился.

Запросы `INSERT` на стороне PostgreSQL выполняются как `COPY "table_name" (field1, field2, ... fieldN) FROM STDIN` внутри PostgreSQL транзакции с автоматическим коммитом после каждого запроса `INSERT`.

PostgreSQL массивы конвертируются в массивы ClickHouse.

## Внимание

Будьте внимательны, в PostgreSQL массивы, созданные как `type_name[]`, являются многомерными и могут содержать в себе разное количество измерений в разных строках одной таблицы. Внутри ClickHouse допустимы только многомерные массивы с одинаковым количеством измерений во всех строках таблицы.

Поддерживает несколько реплик, которые должны быть перечислены через |. Например:

```
CREATE TABLE test_replicas (id UInt32, name String) ENGINE = PostgreSQL(`postgres{2|3|4}:5432`, 'clickhouse', 'test_replicas', 'postgres', 'mysecretpassword');
```

При использовании словаря PostgreSQL поддерживается приоритет реплик. Чем больше номер реплики, тем ниже ее приоритет. Наивысший приоритет у реплики с номером 0.

В примере ниже реплика `example01-1` имеет более высокий приоритет:

```
<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>
```

## Пример использования

Таблица в 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

postgresql> SELECT * FROM test;
 int_id | int_nullable | float | str  | float_nullable
-----+-----+-----+-----+
  1    |      |     2 | test | 
(1 row)

```

Таблица в ClickHouse, получение данных из PostgreSQL таблицы, созданной выше:

```

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');

```

## См. также

- [Табличная функция postgresql](#)
- [Использование PostgreSQL в качестве источника для внешнего словаря](#)

## ExternalDistributed

Движок ExternalDistributed позволяет выполнять запросы SELECT для таблиц на удаленном сервере MySQL или PostgreSQL. Принимает в качестве аргумента табличные движки MySQL или PostgreSQL, поэтому возможно шардирование.

## Создание таблицы

```
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');
```

Смотрите подробное описание запроса [CREATE TABLE](#).

Структура таблицы может отличаться от структуры исходной таблицы:

- Имена столбцов должны быть такими же, как в исходной таблице, но можно использовать только некоторые из этих столбцов и в любом порядке.
- Типы столбцов могут отличаться от типов в исходной таблице. ClickHouse пытается [привести](#) значения к типам данных ClickHouse.

### Параметры движка

- `engine` — табличный движок MySQL или PostgreSQL.
- `host:port` — адрес сервера MySQL или PostgreSQL.
- `database` — имя базы данных на сервере.
- `table` — имя таблицы.
- `user` — имя пользователя.
- `password` — пароль пользователя.

## Особенности реализации

Поддерживает несколько реплик, которые должны быть перечислены через |, а шарды — через ,.  
Например:

```
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');
```

При указании реплик для каждого из шардов при чтении выбирается одна из доступных реплик. Если соединиться не удалось, то выбирается следующая реплика, и так для всех реплик. Если попытка соединения не удалась для всех реплик, то сервер ClickHouse снова пытается соединиться с одной из реплик, перебирая их по кругу, и так несколько раз.

Вы можете указать любое количество шардов и любое количество реплик для каждого шарда.

### Смотрите также

- [Табличный движок MySQL](#)
- [Табличный движок PostgreSQL](#)
- [Табличный движок Distributed](#)

## MaterializedPostgreSQL

Создает таблицу ClickHouse с исходным дампом данных таблицы PostgreSQL и запускает процесс репликации, т.е. выполняется применение новых изменений в фоне, как эти изменения происходят в таблице PostgreSQL в удаленной базе данных PostgreSQL.

Если требуется более одной таблицы, вместо движка таблиц рекомендуется использовать движок баз данных [MaterializedPostgreSQL](#) и с помощью настройки [materialized\\_postgresql\\_tables\\_list](#) указывать таблицы, которые нужно реплицировать. Это будет намного лучше с точки зрения нагрузки на процессор, уменьшит количество подключений и количество слотов репликации внутри удаленной базы данных PostgreSQL.

## Создание таблицы

```
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;
```

### Параметры движка

- `host:port` — адрес сервера PostgreSQL.
- `database` — имя базы данных на удалённом сервере.
- `table` — имя таблицы на удалённом сервере.
- `user` — пользователь PostgreSQL.
- `password` — пароль пользователя.

## Требования

1. Настройка `wal_level` должна иметь значение `logical`, параметр `max_replication_slots` должен быть равен по меньшей мере 2 в конфигурационном файле в PostgreSQL.
2. Таблица, созданная с помощью движка `MaterializedPostgreSQL`, должна иметь первичный ключ — такой же, как `replica identity index` (по умолчанию: первичный ключ) таблицы PostgreSQL (смотрите [replica identity index](#)).
3. Допускается только база данных [Atomic](#).

## Виртуальные столбцы

- `_version` — счетчик транзакций. Тип: [UInt64](#).
- `_sign` — метка удаления. Тип: [Int8](#). Возможные значения:
  - 1 — строка не удалена,
  - -1 — строка удалена.

Эти столбцы не нужно добавлять при создании таблицы. Они всегда доступны в `SELECT` запросе. Столбец `_version` равен позиции `LSN` в `WAL`, поэтому его можно использовать для проверки актуальности репликации.

```
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;
```

## Предупреждение

Репликация **TOAST**-значений не поддерживается. Для типа данных будет использоваться значение по умолчанию.

## Движки таблиц для интеграции

Для интеграции с внешними системами ClickHouse предоставляет различные средства, включая движки таблиц. Конфигурирование интеграционных движков осуществляется с помощью запросов `CREATE TABLE` или `ALTER TABLE`, как и для других табличных движков. С точки зрения пользователя, настроенная интеграция выглядит как обычная таблица, но запросы к ней передаются через прокси во внешнюю систему. Этот прозрачный запрос является одним из ключевых преимуществ этого подхода по сравнению с альтернативными методами интеграции, такими как внешние словари или табличные функции, которые требуют использования пользовательских методов запроса при каждом использовании.

Список поддерживаемых интеграций:

- [ODBC](#)
- [JDBC](#)
- [MySQL](#)
- [MongoDB](#)
- [HDFS](#)
- [S3](#)
- [Kafka](#)
- [EmbeddedRocksDB](#)
- [RabbitMQ](#)
- [PostgreSQL](#)

## RabbitMQ

Движок работает с [RabbitMQ](#).

[RabbitMQ](#) позволяет:

- Публиковать/подписываться на потоки данных.
- Обрабатывать потоки по мере их появления.

## Создание таблицы

```

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',
    rabbitmq_exchange_name = 'exchange_name',
    rabbitmq_format = 'data_format'[,]
    [rabbitmq_exchange_type = 'exchange_type',]
    [rabbitmq_routing_key_list = 'key1,key2,...',]
    [rabbitmq_row_delimiter = 'delimiter_symbol',]
    [rabbitmq_schema = "",]
    [rabbitmq_num_consumers = N,]
    [rabbitmq_num_queues = N,]
    [rabbitmq_queue_base = 'queue',]
    [rabbitmq_persistent = 0,]
    [rabbitmq_skip_broken_messages = N,]
    [rabbitmq_max_block_size = N,]
    [rabbitmq_flush_interval_ms = N]

```

Обязательные параметры:

- `rabbitmq_host_port` – адрес сервера (хост:порт). Например: `localhost:5672`.
- `rabbitmq_exchange_name` – имя точки обмена в RabbitMQ.
- `rabbitmq_format` – формат сообщения. Используется такое же обозначение, как и в функции `FORMAT` в SQL, например, `JSONEachRow`. Подробнее см. в разделе [Форматы входных и выходных данных](#).

Дополнительные параметры:

- `rabbitmq_exchange_type` – тип точки обмена в RabbitMQ: `direct`, `fanout`, `topic`, `headers`, `consistent_hash`. По умолчанию: `fanout`.
- `rabbitmq_routing_key_list` – список ключей маршрутизации, через запятую.
- `rabbitmq_rowDelimiter` – символ-разделитель, который завершает сообщение.
- `rabbitmq_schema` – опциональный параметр, необходимый, если используется формат, требующий определения схемы. Например, `Cap'n Proto` требует путь к файлу со схемой и название корневого объекта `schema.capnp:Message`.
- `rabbitmq_num_consumers` – количество потребителей на таблицу. По умолчанию: 1. Укажите больше потребителей, если пропускная способность одного потребителя недостаточна.
- `rabbitmq_num_queues` – количество очередей. По умолчанию: 1. Большее число очередей может сильно увеличить пропускную способность.
- `rabbitmq_queue_base` - настройка для имен очередей. Сценарии использования описаны ниже.
- `rabbitmq_persistent` - флаг, от которого зависит настройка 'durable' для сообщений при запросах `INSERT`. По умолчанию: 0.
- `rabbitmq_skip_broken_messages` – максимальное количество некорректных сообщений в блоке. Если `rabbitmq_skip_broken_messages = N`, то движок отбрасывает N сообщений, которые не получилось обработать. Одно сообщение в точности соответствует одной записи (строке). Значение по умолчанию – 0.
- `rabbitmq_max_block_size`
- `rabbitmq_flush_interval_ms`

Настройки форматов данных также могут быть добавлены в списке RabbitMQ настроек.

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';
```

Конфигурация сервера RabbitMQ добавляется с помощью конфигурационного файла ClickHouse.

Требуемая конфигурация:

```
<rabbitmq>
  <username>root</username>
  <password>clickhouse</password>
</rabbitmq>
```

Дополнительная конфигурация:

```
<rabbitmq>
  <vhost>clickhouse</vhost>
</rabbitmq>
```

## Описание

Запрос `SELECT` не очень полезен для чтения сообщений (за исключением отладки), поскольку каждое сообщение может быть прочитано только один раз. Практичнее создавать потоки реального времени с помощью **материализованных представлений**. Для этого:

1. Создайте потребителя RabbitMQ с помощью движка и рассматривайте его как поток данных.
2. Создайте таблицу с необходимой структурой.
3. Создайте материализованное представление, которое преобразует данные от движка и помещает их в ранее созданную таблицу.

Когда к движку присоединяется материализованное представление, оно начинает в фоновом режиме собирать данные. Это позволяет непрерывно получать сообщения от RabbitMQ и преобразовывать их в необходимый формат с помощью `SELECT`.

У одной таблицы RabbitMQ может быть неограниченное количество материализованных представлений.

Данные передаются с помощью параметров `rabbitmq_exchange_type` и `rabbitmq_routing_key_list`. Может быть не более одной точки обмена на таблицу. Одна точка обмена может использоваться несколькими таблицами: это позволяет выполнять маршрутизацию по нескольким таблицам одновременно.

Параметры точек обмена:

- `direct` - маршрутизация основана на точном совпадении ключей. Пример списка ключей: `key1,key2,key3,key4,key5`. Ключ сообщения может совпадать с одним из них.
- `fanout` - маршрутизация по всем таблицам, где имя точки обмена совпадает, независимо от ключей.
- `topic` - маршрутизация основана на правилах с ключами, разделенными точками. Например: `*.logs, records.*.*.2020, *.2018,*.2019,*.2020`.

- `headers` - маршрутизация основана на совпадении `key=value` с настройкой `x-match=all` или `x-match=any`. Пример списка ключей таблицы: `x-match=all,format=logs,type=report,year=2020`.
- `consistent_hash` - данные равномерно распределяются между всеми связанными таблицами, где имя точки обмена совпадает. Обратите внимание, что этот тип обмена должен быть включен с помощью плагина RabbitMQ: `rabbitmq-plugins enable rabbitmq_consistent_hash_exchange`.

Настройка `rabbitmq_queue_base` может быть использована в следующих случаях:

1. чтобы восстановить чтение из ранее созданных очередей, если оно прекратилось по какой-либо причине, но очереди остались непустыми. Для восстановления чтения из одной конкретной очереди, нужно написать ее имя в `rabbitmq_queue_base` настройку и не указывать настройки `rabbitmq_num_consumers` и `rabbitmq_num_queues`. Чтобы восстановить чтение из всех очередей, которые были созданы для конкретной таблицы, необходимо совпадение следующих настроек: `rabbitmq_queue_base`, `rabbitmq_num_consumers`, `rabbitmq_num_queues`. По умолчанию, если настройка `rabbitmq_queue_base` не указана, будут использованы уникальные для каждой таблицы имена очередей.
2. чтобы объявить одни и те же очереди для разных таблиц, что позволяет создавать несколько параллельных подписчиков на каждую из очередей. То есть обеспечивается лучшая производительность. В данном случае, для таких таблиц также необходимо совпадение настроек: `rabbitmq_num_consumers`, `rabbitmq_num_queues`.
3. чтобы повторно использовать созданные с `durable` настройкой очереди, так как они не удаляются автоматически (но могут быть удалены с помощью любого RabbitMQ CLI).

Для улучшения производительности полученные сообщения группируются в блоки размера `max_insert_block_size`. Если блок не удалось сформировать за `stream_flush_interval_ms` миллисекунд, то данные будут сброшены в таблицу независимо от полноты блока.

Если параметры `rabbitmq_num_consumers` и/или `rabbitmq_num_queues` заданы вместе с параметром `rabbitmq_exchange_type`:

- плагин `rabbitmq-consistent-hash-exchange` должен быть включен.
- свойство `message_id` должно быть определено (уникальное для каждого сообщения/пакета).

При запросах `INSERT` отправляемым сообщениям добавляются метаданные: `messageID` и флаг `republished` - доступны через заголовки сообщений (`headers`).

Для запросов чтения и вставки не должна использоваться одна и та же таблица.

Пример:

```

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();

CREATE MATERIALIZED VIEW consumer TO daily
AS SELECT key, value FROM queue;

SELECT key, value FROM daily ORDER BY key;

```

## Virtual Columns

- `_exchange_name` - имя точки обмена RabbitMQ.
- `_channel_id` - идентификатор канала ChannelID, на котором было получено сообщение.
- `_delivery_tag` - значение DeliveryTag полученного сообщения. Уникально в рамках одного канала.
- `_redelivered` - флаг `redelivered`. (Не равно нулю, если есть возможность, что сообщение было получено более, чем одним каналом.)
- `_message_id` - значение поля `messageID` полученного сообщения. Данное поле непусто, если указано в параметрах при отправке сообщения.
- `_timestamp` - значение поля `timestamp` полученного сообщения. Данное поле непусто, если указано в параметрах при отправке сообщения.

## Специальные движки таблиц

Существует три основные категории движков таблиц:

- [Семейство MergeTree](#) для основного использования.
- [Семейство Log](#) для небольших временных данных.
- [Движки таблиц для интеграции](#).

Остальные движки таблиц уникальны по своему назначению и еще не сгруппированы в семейства, поэтому они помещены в эту специальную категорию.

## Distributed

Движок **Distributed** не хранит данные **самостоятельно**, а позволяет обрабатывать запросы распределённо, на нескольких серверах. Чтение автоматически распараллеливается. При чтении будут использованы индексы таблиц на удалённых серверах, если есть.

Движок Distributed принимает параметры:

- имя кластера в конфигурационном файле сервера
- имя удалённой базы данных
- имя удалённой таблицы
- (не обязательно) ключ шардирования.
- (не обязательно) имя политики, оно будет использоваться для хранения временных файлов для асинхронной отправки

Смотрите также:

- настройка `insert_distributed_sync`
- [MergeTree](#) для примера

Пример:

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

данные будут читаться со всех серверов кластера logs, из таблицы default.hits, расположенной на каждом сервере кластера.

Данные не только читаются, но и частично (настолько, насколько это возможно) обрабатываются на удалённых серверах.

Например, при запросе с GROUP BY, данные будут агрегированы на удалённых серверах, промежуточные состояния агрегатных функций будут отправлены на запросивший сервер; затем данные будут доагрегированы.

Вместо имени базы данных может использоваться константное выражение, возвращающее строку.

Например, currentDatabase().

logs - имя кластера в конфигурационном файле сервера.

Кластеры задаются следующим образом:

```
<remote_servers>
  <logs>
    <shard>
      <!-- Не обязательно. Вес шарда при записи данных. По умолчанию, 1. -->
      <weight>1</weight>
      <!-- Не обязательно. Записывать ли данные только на одну, любую из реплик. По умолчанию, false -
      записывать данные на все реплики. -->
      <internal_replication>false</internal_replication>
      <replica>
        <!-- Не обязательно. Приоритет реплики для балансировки нагрузки (смотрите также настройку
        load_balancing). По умолчанию : 1 (меньшее значение - больший приоритет). -->
        <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>
        <port>9000</port>
      </replica>
    </shard>
  </logs>
</remote_servers>
```

Здесь задан кластер с именем logs, состоящий из двух шардов, каждый из которых состоит из двух реплик.

Шардами называются серверы, содержащие разные части данных (чтобы прочитать все данные, нужно идти на все шарды).

Репликами называются дублирующие серверы (чтобы прочитать данные, можно идти за данными на любую из реплик).

Имя кластера не должно содержать точки.

В качестве параметров для каждого сервера указываются `host`, `port` и, не обязательно, `user`, `password`, `secure`, `compression`:

- `host` - адрес удалённого сервера. Может быть указан домен, или IPv4 или IPv6 адрес. В случае указания домена, при старте сервера делается DNS запрос, и результат запоминается на всё время работы сервера. Если DNS запрос неуспешен, то сервер не запускается. Если вы изменяете DNS-запись, перезапустите сервер.
- `port` - TCP-порт для межсерверного взаимодействия (в конфиге - `tcp_port`, обычно 9000). Не перепутайте с `http_port`.
- `user` - имя пользователя для соединения с удалённым сервером. по умолчанию - `default`. Этот пользователь должен иметь доступ для соединения с указанным сервером. Доступы настраиваются в файле `users.xml`, подробнеесмотрите в разделе [Права доступа](#).
- `password` - пароль для соединения с удалённым сервером, в открытом виде. по умолчанию - пустая строка.
- `secure` - Использовать шифрованное соединение `ssl`, Обычно используется с портом `port = 9440`. Сервер должен слушать порт `<tcp_port_secure>9440</tcp_port_secure>` с корректными настройками сертификатов.
- `compression` - Использовать сжатие данных. По умолчанию: `true`.

При указании реплик, для каждого из шардов, при чтении, будет выбрана одна из доступных реплик. Можно настроить алгоритм балансировки нагрузки (то есть, предпочтения, на какую из реплик идти) - см. настройку [load\\_balancing](#).

Если соединение с сервером не установлено, то будет произведена попытка соединения с небольшим таймаутом. Если соединиться не удалось, то будет выбрана следующая реплика, и так для всех реплик. Если попытка соединения для всех реплик не удалась, то будут снова произведены попытки соединения по кругу, и так несколько раз.

Это работает в пользу отказоустойчивости, хотя и не обеспечивает полную отказоустойчивость: удалённый сервер может принять соединение, но не работать, или плохо работать.

Можно указать от одного шарда (в таком случае, обработку запроса стоит называть удалённой, а не распределённой) до произвольного количества шардов. В каждом шарде можно указать от одной до произвольного числа реплик. Можно указать разное число реплик для каждого шарда.

Вы можете прописать сколько угодно кластеров в конфигурации.

Для просмотра имеющихся кластеров, вы можете использовать системную таблицу `system.clusters`.

Движок `Distributed` позволяет работать с кластером, как с локальным сервером. При этом, кластер является неэластичным: вы должны прописать его конфигурацию в конфигурационный файл сервера (лучше всех серверов кластера).

Как видно, движок `Distributed` требует прописывания кластера в конфигурационный файл; кластера из конфигурационного файла обновляются налету, без перезапуска сервера. Если вам необходимо каждый раз отправлять запрос на неизвестный набор шардов и реплик, вы можете не создавать `Distributed` таблицу, а воспользоваться табличной функцией `remote`. Смотрите раздел [Табличные функции](#).

Есть два способа записывать данные на кластер:

Во-первых, вы можете самостоятельно определять, на какие серверы какие данные записывать, и выполнять запись непосредственно на каждый шард. То есть, делать `INSERT` в те таблицы, на которые «смотрит» распределённая таблица. Это наиболее гибкое решение поскольку вы можете использовать любую схему шардирования, которая может быть нетривиальной из-за требований предметной области.

Также это является наиболее оптимальным решением, так как данные могут записываться на разные шарды полностью независимо.

Во-вторых, вы можете делать INSERT в Distributed таблицу. В этом случае, таблица будет сама распределять вставляемые данные по серверам. Для того, чтобы писать в Distributed таблицу, у неё должен быть задан ключ шардирования (последний параметр). Также, если шард всего-лишь один, то запись работает и без указания ключа шардирования (так как в этом случае он не имеет смысла).

У каждого шарда в конфигурационном файле может быть задан «вес» (weight). По умолчанию, вес равен единице. Данные будут распределяться по шардам в количестве, пропорциональном весу шарда. Например, если есть два шарда, и у первого выставлен вес 9, а у второго 10, то на первый будет отправляться 9 / 19 доля строк, а на второй - 10 / 19.

У каждого шарда в конфигурационном файле может быть указан параметр internal\_replication.

Если он выставлен в true, то для записи будет выбираться первая живая реплика и данные будут писаться на неё. Этот вариант следует использовать, если Distributed таблица «смотрит» на реплицируемые таблицы. То есть, если таблица, в которую будут записаны данные, будет сама заниматься их репликацией.

Если он выставлен в false (по умолчанию), то данные будут записываться на все реплики. По сути, это означает, что Distributed таблица занимается репликацией данных самостоятельно. Это хуже, чем использование реплицируемых таблиц, так как не контролируется консистентность реплик, и они со временем будут содержать немного разные данные.

Для выбора шарда, на который отправляется строка данных, вычисляется выражение шардирования, и берётся его остаток от деления на суммарный вес шардов. Стока отправляется на шард, соответствующий полуинтервалу остатков от prev\_weights до prev\_weights + weight, где prev\_weights - сумма весов шардов с меньшим номером, а weight - вес этого шарда. Например, если есть два шарда, и у первого выставлен вес 9, а у второго 10, то строка будет отправляться на первый шард для остатков из диапазона [0, 9), а на второй - для остатков из диапазона [9, 19].

Выражением шардирование может быть произвольное выражение от констант и столбцов таблицы, возвращающее целое число. Например, вы можете использовать выражение rand() для случайного распределения данных, или UserID - для распределения по остатку от деления идентификатора посетителя (тогда данные одного посетителя будут расположены на одном шарде, что упростит выполнение IN и JOIN по посетителям). Если распределение какого-либо столбца недостаточно равномерное, вы можете обернуть его в хэш функцию: intHash64(UserID).

Простой остаток от деления является довольно ограниченным решением для шардирования и подходит не для всех случаев. Он подходит для среднего и большого объёма данных (десятки серверов), но не для очень больших объёмов данных (сотни серверов и больше). В последнем случае, лучше использовать схему шардирования, продиктованную требованиями предметной области, и не использовать возможность записи в Distributed таблицы.

Запросы SELECT отправляются на все шарды, и работают независимо от того, каким образом данные распределены по шардам (они могут быть распределены полностью случайно). При добавлении нового шарда, можно не переносить на него старые данные, а записывать новые данные с большим весом - данные будут распределены слегка неравномерно, но запросы будут работать корректно и достаточно эффективно.

Беспокоиться о схеме шардирования имеет смысл в следующих случаях:

- используются запросы, требующие соединение данных (IN, JOIN) по определённому ключу - тогда если данные шардированы по этому ключу, то можно использовать локальные IN, JOIN вместо GLOBAL IN, GLOBAL JOIN, что кардинально более эффективно.
- используется большое количество серверов (сотни и больше) и большое количество маленьких запросов (запросы отдельных клиентов - сайтов, рекламодателей, партнёров) - тогда, для того, чтобы маленькие запросы не затрагивали весь кластер, имеет смысл располагать данные одного клиента на одном шарде, или (вариант, который используется в Яндекс.Метрике) сделать двухуровневое шардирование: разбить весь кластер на «слои», где слой может состоять из нескольких шардов; данные для одного клиента располагаются на одном слое, но в один слой можно по мере необходимости добавлять шарды, в рамках которых данные распределены произвольным образом; создаются распределённые таблицы на каждый слой и одна общая распределённая таблица для глобальных запросов.

Запись данных осуществляется полностью асинхронно. При вставке в таблицу, блок данных сначала записывается в файловую систему. Затем, в фоновом режиме отправляются на удалённые серверы при первой возможности. Период отправки регулируется настройками `distributed_directory_monitor_sleep_time_ms` и `distributed_directory_monitor_max_sleep_time_ms`.

Движок таблиц Distributed отправляет каждый файл со вставленными данными отдельно, но можно включить пакетную отправку данных настройкой `distributed_directory_monitor_batch_inserts`. Эта настройка улучшает производительность кластера за счет более оптимального использования ресурсов сервера-отправителя и сети. Необходимо проверять, что данные отправлены успешно, для этого проверьте список файлов (данных, ожидающих отправки) в каталоге таблицы `/var/lib/clickhouse/data/database/table/`. Количество потоков для выполнения фоновых задач можно задать с помощью настройки `background_distributed_schedule_pool_size`.

Если после INSERT-а в Distributed таблицу, сервер перестал существовать или был грубо перезапущен (например, в следствие аппаратного сбоя), то записанные данные могут быть потеряны. Если в директории таблицы обнаружен повреждённый кусок данных, то он переносится в поддиректорию `broken` и больше не используется.

При выставлении опции `max_parallel_replicas` выполнение запроса распараллеливается по всем репликам внутри одного шарда. Подробнее смотрите раздел [max\\_parallel\\_replicas](#).

## Виртуальные столбцы

- `_shard_num` — содержит значение `shard_num` из таблицы `system.clusters`. Тип: [UInt32](#).

### Примечание

Так как табличные функции `remote` и `cluster` создают временную таблицу на движке `Distributed`, то в ней также доступен столбец `_shard_num`.

### См. также

- общее описание [виртуальных столбцов](#)
- настройка `background_distributed_schedule_pool_size`
- функции `shardNum()` и `shardCount()`

## Dictionary

Движок Dictionary отображает данные [словаря](#) как таблицу ClickHouse.

Рассмотрим для примера словарь products со следующей конфигурацией:

```
<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>
```

Запрос данных словаря:

```
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

В таком виде данные из словаря можно получить при помощи функций `dictGet*`.

Такое представление неудобно, когда нам необходимо получить данные в чистом виде, а также при выполнении операции `JOIN`. Для этих случаев можно использовать движок `Dictionary`, который отобразит данные словаря в таблицу.

Синтаксис:

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

Пример использования:

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

Проверим что у нас в таблице?

```
select * from products limit 1;
```

product_id	title
152689	Some item

## Смотрите также

- [Функция dictionary](#)

## Merge

Движок Merge (не путайте с движком MergeTree) не хранит данные самостоятельно, а позволяет читать одновременно из произвольного количества других таблиц.

Чтение автоматически распараллеливается. Запись в таблицу не поддерживается. При чтении будут использованы индексы тех таблиц, из которых реально идёт чтение, если они существуют.

Движок Merge принимает параметры: имя базы данных и регулярное выражение для таблиц.

Пример:

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

Данные будут читаться из таблиц в базе hits, имена которых соответствуют регулярному выражению '^WatchLog'.

Вместо имени базы данных может использоваться константное выражение, возвращающее строку. Например, `currentDatabase()`.

Регулярные выражения — [re2](#) (поддерживает подмножество PCRE), регистрозависимые.

Смотрите замечание об экранировании в регулярных выражениях в разделе «match».

При выборе таблиц для чтения, сама Merge-таблица не будет выбрана, даже если попадает под регулярное выражение, чтобы не возникло циклов.

Впрочем, вы можете создать две Merge-таблицы, которые будут пытаться бесконечно читать данные друг друга, но делать этого не нужно.

Типичный способ использования движка Merge — работа с большим количеством таблиц типа TinyLog, как с одной.

Пример 2:

Пусть есть старая таблица WatchLog\_old. Необходимо изменитьpartitionирование без перемещения данных в новую таблицу WatchLog\_new. При этом в выборке должны участвовать данные обеих таблиц.

```

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
2018-01-02	2	hit	3

## Виртуальные столбцы

- `_table` — содержит имя таблицы, из которой данные были прочитаны. Тип — [String](#).

В секции `WHERE/PREWHERE` можно установить константное условие на столбец `_table` (например, `WHERE \_table='xyz'`). В этом случае операции чтения выполняются только для тех таблиц, для которых выполняется условие на значение `_table`, таким образом, столбец `_table` работает как индекс.

### Смотрите также

- [Виртуальные столбцы](#)

## File(Format)

Управляет данными в одном файле на диске в указанном формате.

Примеры применения:

- Выгрузка данных из ClickHouse в файл.
- Преобразование данных из одного формата в другой.
- Обновление данных в ClickHouse редактированием файла на диске.

## Использование движка в сервере ClickHouse

### File(Format)

Format должен быть таким, который ClickHouse может использовать и в запросах `INSERT` и в запросах `SELECT`. Полный список поддерживаемых форматов смотрите в разделе [Форматы](#).

Сервер ClickHouse не позволяет указать путь к файлу, с которым будет работать `File`. Используется путь к хранилищу, определенный параметром `path` в конфигурации сервера.

При создании таблицы с помощью `File(Format)` сервер ClickHouse создает в хранилище каталог с именем таблицы, а после добавления в таблицу данных помещает туда файл `data.Format`.

Можно вручную создать в хранилище каталог таблицы, поместить туда файл, затем на сервере ClickHouse добавить ([ATTACH](#)) информацию о таблице, соответствующей имени каталога и прочитать из файла данные.

## Warning

Будьте аккуратны с этой функциональностью, поскольку сервер ClickHouse не отслеживает внешние изменения данных. Если в файл будет производиться запись одновременно со стороны сервера ClickHouse и с внешней стороны, то результат непредсказуем.

### Пример:

1. Создадим на сервере таблицу file\_engine\_table:

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

В конфигурации по умолчанию сервер ClickHouse создаст каталог /var/lib/clickhouse/data/default/file\_engine\_table.

2. Вручную создадим файл /var/lib/clickhouse/data/default/file\_engine\_table/data.TabSeparated с содержимым:

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

3. Запросим данные:

```
SELECT * FROM file_engine_table
```

name	value
one	1
two	2

## Использование движка в Clickhouse-local

В **clickhouse-local** движок в качестве параметра принимает не только формат, но и путь к файлу. В том числе можно указать стандартные потоки ввода/вывода цифровым или буквенным обозначением `0` или `stdin`, `1` или `stdout`. Можно записывать и читать сжатые файлы. Для этого нужно задать дополнительный параметр движка или расширение файла (`gz`, `br` или `xz`).

### Пример:

```
$ 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"
```

## Детали реализации

- Поддерживается одновременное выполнение множества запросов `SELECT`, запросы `INSERT` могут выполняться только последовательно.
- Поддерживается создание ещё не существующего файла при запросе `INSERT`.
- Для существующих файлов `INSERT` записывает в конец файла.

- Не поддерживается:
  - использование операций `ALTER` и `SELECT...SAMPLE`;
  - индексы;
  - репликация.

При записи в таблицу типа Null, данные игнорируются. При чтении из таблицы типа Null, возвращается пустота.

Тем не менее, есть возможность создать материализованное представление над таблицей типа Null. Тогда данные, записываемые в таблицу, будут попадать в представление.

## Set

Представляет собой множество, постоянно находящееся в оперативке. Предназначено для использования в правой части оператора IN (смотрите раздел «Операторы IN»).

В таблицу можно вставлять данные INSERT-ом - будут добавлены новые элементы в множество, с игнорированием дубликатов.

Но из таблицы нельзя, непосредственно, делать SELECT. Единственная возможность чтения - использование в правой части оператора IN.

Данные постоянно находятся в оперативке. При INSERT-е, в директорию таблицы на диске, также пишутся блоки вставленных данных. При запуске сервера, эти данныечитываются в оперативку. То есть, после перезапуска, данные остаются на месте.

При грубом перезапуске сервера, блок данных на диске может быть потерян или повреждён. В последнем случае, может потребоваться вручную удалить файл с повреждёнными данными.

## Ограничения и настройки

При создании таблицы, применяются следующие параметры:

- `persistent`

## Join

Подготовленная структура данных для использования в операциях `JOIN`.

## Создание таблицы

```
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, ...])
```

Смотрите подробное описание запроса `CREATE TABLE`.

### Параметры движка

- `join_strictness` – `строгость JOIN`.
- `join_type` – `тип JOIN`.
- `k1[, k2, ...]` – ключевые столбцы секции `USING` с которыми выполняется операция `JOIN`.

Вводите параметры `join_strictness` и `join_type` без кавычек, например, `Join(ANY, LEFT, col1)`. Они должны быть такими же как и в той операции `JOIN`, в которой таблица будет использоваться. Если параметры не совпадают, ClickHouse не генерирует исключение и может возвращать неверные данные.

## Особенности и рекомендации

### Хранение данных

Данные таблиц `Join` всегда находятся в оперативной памяти. При вставке строк в таблицу ClickHouse записывает блоки данных в каталог на диске, чтобы их можно было восстановить при перезапуске сервера.

При аварийном перезапуске сервера блок данных на диске может быть потерян или повреждён. В последнем случае может потребоваться вручную удалить файл с повреждёнными данными.

### Выборка и добавление данных

Для добавления данных в таблицы с движком `Join` используйте запрос `INSERT`. Если таблица создавалась со строгостью `ANY`, то данные с повторяющимися ключами игнорируются. Если задавалась строгость `ALL`, то добавляются все строки.

Основные применения `Join` таблиц:

- Использование в правой части секции `JOIN`.
- Извлечение данных из таблицы таким же образом как из словаря с помощью функции `joinGet`.

### Удаление данных

Запросы `ALTER DELETE` для таблиц с движком `Join` выполняются как **мутации**. При выполнении мутации `DELETE` считаются отфильтрованные данные и перезаписываются в оперативную память и на диск.

### Ограничения и настройки

При создании таблицы применяются следующие настройки:

- `join_use_nulls`
- `max_rows_in_join`
- `max_bytes_in_join`
- `join_overflow_mode`
- `join_any_take_last_row`
- `persistent`

Таблицы с движком `Join` нельзя использовать в операциях `GLOBAL JOIN`.

Движок `Join` позволяет использовать настройку `join_use_nulls` в запросе `CREATE TABLE`. Необходимо использовать одно и то же значение параметра `join_use_nulls` в запросах `CREATE TABLE` и `SELECT`.

## Примеры использования

Создание левой таблицы:

```
CREATE TABLE id_val(`id` UInt32, `val` UInt32) ENGINE = TinyLog;
```

```
INSERT INTO id_val VALUES (1,11)(2,12)(3,13);
```

Создание правой таблицы с движком Join:

```
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);
```

Объединение таблиц:

```
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

В качестве альтернативы, можно извлечь данные из таблицы Join, указав значение ключа объединения:

```
SELECT joinGet('id_val_join', 'val', toUInt32(1));
```

```
joinGet('id_val_join', 'val', toUInt32(1))  
21 |
```

Удаление данных из таблицы Join:

```
ALTER TABLE id_val_join DELETE WHERE id = 3;
```

id	val
1	21

## URL(URL, Format)

Управляет данными на удаленном HTTP/HTTPS сервере. Данный движок похож на движок [File](#).

### Использование движка в сервере ClickHouse

Format должен быть таким, который ClickHouse может использовать в запросах `SELECT` и, если есть необходимость, `INSERT`. Полный список поддерживаемых форматов смотрите в разделе [Форматы](#).

URL должен соответствовать структуре Uniform Resource Locator. По указанному URL должен находиться сервер работающий по протоколу HTTP или HTTPS. При этом не должно требоваться никаких дополнительных заголовков для получения ответа от сервера.

Запросы INSERT и SELECT транслируются в POST и GET запросы соответственно. Для обработки POST-запросов удаленный сервер должен поддерживать **Chunked transfer encoding**.

Максимальное количество переходов по редиректам при выполнении HTTP-запроса методом GET можно ограничить с помощью настройки **max\_http\_get\_redirects**.

### Пример:

1. Создадим на сервере таблицу url\_engine\_table:

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

2. Создадим простейший http-сервер стандартными средствами языка python3 и запустим его:

```
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. Запросим данные:

```
SELECT * FROM url_engine_table
```

word	value
Hello	1
World	2

## Особенности использования

- Поддерживается многопоточное чтение и запись.
- Не поддерживается:
  - использование операций ALTER и SELECT...SAMPLE;
  - индексы;
  - репликация.

## View

Используется для реализации представлений (подробнее см. запрос [CREATE VIEW](#)). Не хранит данные, а хранит только указанный запрос [SELECT](#). При чтении из таблицы, выполняет его (с удалением из запроса всех ненужных столбцов).

## MaterializedView

Используется для реализации материализованных представлений (подробнее см. запрос [CREATE TABLE](#)). Для хранения данных, использует другой движок, который был указан при создании представления. При чтении из таблицы, просто использует этот движок.

## Memory

Хранит данные в оперативке, в несжатом виде. Данные хранятся именно в таком виде, в каком они получаются при чтении. То есть, само чтение из этой таблицы полностью бесплатно.

Конкурентный доступ к данным синхронизируется. Блокировки короткие: чтения и записи не блокируют друг друга.

Индексы не поддерживаются. Чтение распараллеливается.

За счёт отсутствия чтения с диска, разжатия и десериализации данных удаётся достичь максимальной производительности (выше 10 ГБ/сек.) на простых запросах. (Стоит заметить, что во многих случаях, производительность движка MergeTree, почти такая же высокая.)

При перезапуске сервера данные из таблицы исчезают и таблица становится пустой.

Обычно, использование этого движка таблиц является неоправданным. Тем не менее, он может использоваться для тестов, а также в задачах, где важно достичь максимальной скорости на не очень большом количестве строк (примерно до 100 000 000).

Движок Memory используется системой для временных таблиц - внешних данных запроса (смотрите раздел «Внешние данные для обработки запроса»), для реализации [GLOBAL IN](#) (смотрите раздел «Операторы IN»).

## Buffer

Буферизует записываемые данные в оперативке, периодически сбрасывая их в другую таблицу. При чтении, производится чтение данных одновременно из буфера и из другой таблицы.

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

Параметры движка:

`database` — имя базы данных. Вместо имени базы данных может использоваться константное выражение, возвращающее строку.

`table` — таблица, в которую сбрасывать данные.

`num_layers` — уровень параллелизма. Физически таблица будет представлена в виде `num_layers` независимых буферов. Рекомендуемое значение — 16.

`min_time, max_time, min_rows, max_rows, min_bytes, max_bytes` — условия для сброса данных из буфера.

Данные сбрасываются из буфера и записываются в таблицу назначения, если выполнены все `min`-условия или хотя бы одно `max`-условие.

- `min_time, max_time` — условие на время в секундах от момента первой записи в буфер.
- `min_rows, max_rows` — условие на количество строк в буфере.
- `min_bytes, max_bytes` — условие на количество байт в буфере.

При записи, данные вставляются в случайный из `num_layers` буферов. Или, если размер куска вставляемых данных достаточно большой (больше `max_rows` или `max_bytes`), то он записывается в таблицу назначения минуя буфер.

Условия для сброса данных учитываются отдельно для каждого из `num_layers` буферов. Например, если `num_layers = 16` и `max_bytes = 100000000`, то максимальный расход оперативки будет 1.6 GB.

Пример:

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

Создаём таблицу `merge.hits_buffer` такой же структуры как `merge.hits` и движком `Buffer`. При записи в эту таблицу, данные буферизуются в оперативке и, в дальнейшем, записываются в таблицу `merge.hits`. Создаётся 16 буферов. Данные, имеющиеся в каждом из них будут сбрасываться, если прошло сто секунд, или записан миллион строк, или записано сто мегабайт данных; или если одновременно прошло десять секунд и записано десять тысяч строк и записано десять мегабайт данных. Для примера, если записана всего лишь одна строка, то через сто секунд она будет сброшена в любом случае. А если записано много строк, то они будут сброшены раньше.

При остановке сервера, при `DROP TABLE` или `DETACH TABLE`, данные из буфера тоже сбрасываются в таблицу назначения.

В качестве имени базы данных и имени таблицы можно указать пустые строки в одинарных кавычках. Это обозначает отсутствие таблицы назначения. В таком случае, при достижении условий на сброс данных, буфер будет просто очищаться. Это может быть полезным, чтобы хранить в оперативке некоторое окно данных.

При чтении из таблицы типа `Buffer`, будут обработаны данные, как находящиеся в буфере, так и данные из таблицы назначения (если такая есть).

Но следует иметь ввиду, что таблица `Buffer` не поддерживает индекс. То есть, данные в буфере будут просканированы полностью, что может быть медленно для буферов большого размера. (Для данных в подчинённой таблице, будет использоваться тот индекс, который она поддерживает.)

Если множество столбцов таблицы `Buffer` не совпадает с множеством столбцов подчинённой таблицы, то будут вставлено подмножество столбцов, которое присутствует в обеих таблицах.

Если у одного из столбцов таблицы `Buffer` и подчинённой таблицы не совпадает тип, то в лог сервера будет записано сообщение об ошибке и буфер будет очищен.

То же самое происходит, если подчинённая таблица не существует в момент сброса буфера.

Если есть необходимость выполнить `ALTER` для подчинённой таблицы и для таблицы `Buffer`, то рекомендуется удалить таблицу `Buffer`, затем выполнить `ALTER` подчинённой таблицы, а после создать таблицу `Buffer` заново.

## Внимание

В релизах до 28 сентября 2020 года выполнение `ALTER` на таблице `Buffer` ломает структуру блоков и вызывает ошибку (см. [#15117](#)), поэтому удаление буфера и его пересоздание — единственный вариант миграции для данного движка. Перед выполнением `ALTER` на таблице `Buffer` убедитесь, что в вашей версии эта ошибка устранена.

При непредвиденном перезапуске сервера, данные, находящиеся в буфере, будут потеряны.

Для таблиц типа Buffer неправильно работают FINAL и SAMPLE. Эти условия пробрасываются в таблицу назначения, но не используются для обработки данных в буфере. В связи с этим, рекомендуется использовать таблицу типа Buffer только для записи, а читать из таблицы назначения.

При добавлении данных в Buffer, один из буферов блокируется. Это приводит к задержкам, если одновременно делается чтение из таблицы.

Данные, вставляемые в таблицу Buffer, попадают в подчинённую таблицу в порядке, возможно отличающимся от порядка вставки, и блоками, возможно отличающимися от вставленных блоков. В связи с этим, трудно корректно использовать таблицу типа Buffer для записи в CollapsingMergeTree. Чтобы избежать проблем, можно выставить num\_layers в 1.

Если таблица назначения является реплицируемой, то при записи в таблицу Buffer будут потеряны некоторые ожидаемые свойства реплицируемых таблиц. Из-за произвольного изменения порядка строк и размеров блоков данных, перестаёт работать дедупликация данных, в результате чего исчезает возможность надёжной exactly once записи в реплицируемые таблицы.

В связи с этими недостатками, таблицы типа Buffer можно рекомендовать к применению лишь в очень редких случаях.

Таблицы типа Buffer используются в тех случаях, когда от большого количества серверов поступает слишком много INSERT-ов в единицу времени, и нет возможности заранее самостоятельно буферизовать данные перед вставкой, в результате чего, INSERT-ы не успевают выполняться.

Заметим, что даже для таблиц типа Buffer не имеет смысла вставлять данные по одной строке, так как таким образом будет достигнута скорость всего лишь в несколько тысяч строк в секунду, тогда как при вставке более крупными блоками, достижимо более миллиона строк в секунду (смотрите раздел «Производительность»).

## Внешние данные для обработки запроса

ClickHouse позволяет отправить на сервер данные, необходимые для обработки одного запроса, вместе с запросом SELECT. Такие данные будут положены во временную таблицу (см. раздел «Временные таблицы») и смогут использоваться в запросе (например, в операторах IN).

Для примера, если у вас есть текстовый файл с важными идентификаторами посетителей, вы можете загрузить его на сервер вместе с запросом, в котором используется фильтрация по этому списку.

Если вам нужно будет выполнить более одного запроса с достаточно большими внешними данными - лучше не использовать эту функциональность, а загрузить данные в БД заранее.

Внешние данные могут быть загружены как с помощью клиента командной строки (в не интерактивном режиме), так и через HTTP-интерфейс.

В клиенте командной строки, может быть указана секция параметров вида

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

Таких секций может быть несколько - по числу передаваемых таблиц.

**-external** - маркер начала секции.

**-file** - путь к файлу с дампом таблицы, или -, что обозначает stdin.

Из stdin может быть считана только одна таблица.

Следующие параметры не обязательные:

**-name** - имя таблицы. Если не указано - используется \_data.

**-format** - формат данных в файле. Если не указано - используется TabSeparated.

Должен быть указан один из следующих параметров:

**-types** - список типов столбцов через запятую. Например, UInt64,String. Столбцы будут названы \_1, \_2, ...

**-structure** - структура таблицы, в форме UserID UInt64, URL String. Определяет имена и типы столбцов.

Файлы, указанные в file, будут разобраны форматом, указанным в format, с использованием типов данных, указанных в types или structure. Таблица будет загружена на сервер, и доступна там в качестве временной таблицы с именем name.

Примеры:

```
$ 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
```

При использовании HTTP интерфейса, внешние данные передаются в формате multipart/form-data. Каждая таблица передаётся отдельным файлом. Имя таблицы берётся из имени файла. В query\_string передаются параметры name\_format, name\_types, name\_structure, где name - имя таблицы, которой соответствуют эти параметры. Смысл параметров такой же, как при использовании клиента командной строки.

Пример:

```
$ 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
```

При распределённой обработке запроса, временные таблицы передаются на все удалённые серверы.

## 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

## Движки баз данных

Движки баз данных обеспечивают работу с таблицами.

По умолчанию ClickHouse использует движок [Atomic](#). Он поддерживает конфигурируемые [движки таблиц](#) и [диалект SQL](#).

Также можно использовать следующие движки баз данных:

- [MySQL](#)
- [MaterializedMySQL](#)
- [Lazy](#)
- [PostgreSQL](#)
- [Replicated](#)

# [экспериментальный] MaterializedMySQL

**Это экспериментальный движок, который не следует использовать в продакшене.**

Создает базу данных ClickHouse со всеми таблицами, существующими в MySQL, и всеми данными в этих таблицах.

Сервер ClickHouse работает как реплика MySQL. Он читает файл binlog и выполняет DDL and DML-запросы.

## Создание базы данных

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

### Параметры движка

- `host:port` — адрес сервера MySQL.
- `database` — имя базы данных на удалённом сервере.
- `user` — пользователь MySQL.
- `password` — пароль пользователя.

### Настройки движка

- `max_rows_in_buffer` — максимальное количество строк, содержимое которых может кешироваться в памяти (для одной таблицы и данных кеша, которые невозможно запросить). При превышении количества строк, данные будут материализованы. Значение по умолчанию: 65 505.
- `max_bytes_in_buffer` — максимальное количество байтов, которое разрешено кешировать в памяти (для одной таблицы и данных кеша, которые невозможно запросить). При превышении количества строк, данные будут материализованы. Значение по умолчанию: 1 048 576.
- `max_rows_in_buffers` — максимальное количество строк, содержимое которых может кешироваться в памяти (для базы данных и данных кеша, которые невозможно запросить). При превышении количества строк, данные будут материализованы. Значение по умолчанию: 65 505.
- `max_bytes_in_buffers` — максимальное количество байтов, которое разрешено кешировать данным в памяти (для базы данных и данных кеша, которые невозможно запросить). При превышении количества строк, данные будут материализованы. Значение по умолчанию: 1 048 576.
- `max_flush_data_time` — максимальное время в миллисекундах, в течение которого разрешено кешировать данные в памяти (для базы данных и данных кеша, которые невозможно запросить). При превышении количества указанного периода, данные будут материализованы. Значение по умолчанию: 1000.
- `max_wait_time_when_mysql_unavailable` — интервал между повторными попытками, если MySQL недоступен. Указывается в миллисекундах. Отрицательное значение отключает повторные попытки. Значение по умолчанию: 1000.
- `allows_query_when_mysql_lost` — признак, разрешен ли запрос к материализованной таблице при потере соединения с MySQL. Значение по умолчанию: 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;
```

## Настройки на стороне MySQL-сервера

Для правильной работы MaterializedMySQL следует обязательно указать на сервере MySQL следующие параметры конфигурации:

- `default_authentication_plugin = mysql_native_password` — MaterializedMySQL может авторизоваться только с помощью этого метода.
- `gtid_mode = on` — ведение журнала на основе GTID является обязательным для обеспечения правильной репликации.

## Внимание

При включении `gtid_mode` вы также должны указать `enforce_gtid_consistency = on`.

## Виртуальные столбцы

При работе с движком баз данных MaterializedMySQL используются таблицы семейства [ReplacingMergeTree](#) с виртуальными столбцами `_sign` и `_version`.

- `_version` — счетчик транзакций. Тип [UInt64](#).
- `_sign` — метка удаления. Тип [Int8](#). Возможные значения:
  - 1 — строка не удалена,
  - -1 — строка удалена.

## Поддержка типов данных

MySQL	ClickHouse
TINY	<a href="#">Int8</a>
SHORT	<a href="#">Int16</a>
INT24	<a href="#">Int32</a>
LONG	<a href="#">UInt32</a>
LONGLONG	<a href="#">UInt64</a>
FLOAT	<a href="#">Float32</a>
DOUBLE	<a href="#">Float64</a>
DECIMAL, NEWDECIMAL	<a href="#">Decimal</a>
DATE, NEWDATE	<a href="#">Date</a>
DATETIME, TIMESTAMP	<a href="#">DateTime</a>

MySQL	ClickHouse
DATETIME2, TIMESTAMP2	DateTime64
ENUM	Enum
STRING	String
VARCHAR, VAR_STRING	String
BLOB	String
BINARY	FixedString

Тип `Nullable` поддерживается.

Другие типы не поддерживаются. Если таблица MySQL содержит столбец другого типа, ClickHouse выдаст исключение "Неподдерживаемый тип данных" ("Unhandled data type") и остановит репликацию.

## Особенности и рекомендации

### Ограничения совместимости

Кроме ограничений на типы данных, существует несколько ограничений по сравнению с базами данных MySQL, которые следует решить до того, как станет возможной репликация:

- Каждая таблица в MySQL должна содержать `PRIMARY KEY`.
- Репликация для таблиц, содержащих строки со значениями полей `ENUM` вне диапазона значений (определяется размерностью `ENUM`), не будет работать.

### DDL-запросы

DDL-запросы в MySQL конвертируются в соответствующие DDL-запросы в ClickHouse (`ALTER`, `CREATE`, `DROP`, `RENAME`). Если ClickHouse не может конвертировать какой-либо DDL-запрос, он его игнорирует.

### Репликация данных

Данные являются неизменяемыми со стороны пользователя ClickHouse, но автоматически обновляются путём репликации следующих запросов из MySQL:

- Запрос `INSERT` конвертируется в ClickHouse в `INSERT` с `_sign=1`.
- Запрос `DELETE` конвертируется в ClickHouse в `INSERT` с `_sign=-1`.
- Запрос `UPDATE` конвертируется в ClickHouse в `INSERT` с `_sign=-1` и `INSERT` с `_sign=1`.

### Выборка из таблиц движка MaterializedMySQL

Запрос `SELECT` из таблиц движка `MaterializedMySQL` имеет некоторую специфику:

- Если в запросе `SELECT` напрямую не указан столбец `_version`, то используется модификатор `FINAL`. Таким образом, выбираются только строки с `MAX(_version)`.
- Если в запросе `SELECT` напрямую не указан столбец `_sign`, то по умолчанию используется `WHERE _sign=1`. Таким образом, удаленные строки не включаются в результирующий набор.

- Результат включает комментарии к столбцам, если они существуют в таблицах базы данных MySQL.

## Конвертация индексов

Секции PRIMARY KEY и INDEX в MySQL конвертируются в кортежи ORDER BY в таблицах ClickHouse.

В таблицах ClickHouse данные физически хранятся в том порядке, который определяется секцией ORDER BY. Чтобы физически перегруппировать данные, используйте [материализованные представления](#).

### Примечание

- Строки с `_sign=-1` физически не удаляются из таблиц.
- Каскадные запросы `UPDATE/DELETE` не поддерживаются движком `MaterializedMySQL`.
- Репликация может быть легко нарушена.
- Прямые операции изменения данных в таблицах и базах данных `MaterializedMySQL` запрещены.
- На работу `MaterializedMySQL` влияет настройка `optimize_on_insert`. Когда таблица на MySQL сервере меняется, происходит слияние данных в соответствующей таблице в базе данных `MaterializedMySQL`.

## Примеры использования

Запросы в 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!

База данных в ClickHouse, обмен данными с сервером MySQL:

База данных и созданная таблица:

```
CREATE DATABASE mysql ENGINE = MaterializedMySQL('localhost:3306', 'db', 'user', '***');
SHOW TABLES FROM mysql;
```

name
test

После вставки данных:

```
SELECT * FROM mysql.test;
```

a	b
1	11
2	22

После удаления данных, добавления столбца и обновления:

```
SELECT * FROM mysql.test;
```

a	b	c
2	222	Wow!

## [экспериментальный] MaterializedPostgreSQL

Создает базу данных ClickHouse с исходным дампом данных таблиц PostgreSQL и запускает процесс репликации, т.е. выполняется применение новых изменений в фоне, как эти изменения происходят в таблице PostgreSQL в удаленной базе данных PostgreSQL.

Сервер ClickHouse работает как реплика PostgreSQL. Он читает WAL и выполняет DML запросы. Данные, полученные в результате DDL запросов, не реплицируются, но сами запросы могут быть обработаны (описано ниже).

### Создание базы данных

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

#### Параметры движка

- `host:port` — адрес сервера PostgreSQL.
- `database` — имя базы данных на удалённом сервере.
- `user` — пользователь PostgreSQL.
- `password` — пароль пользователя.

### Настройки

- `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;
```

# Требования

- Настройка `wal_level` должна иметь значение `logical`, параметр `max_replication_slots` должен быть равен по меньшей мере 2 в конфигурационном файле в PostgreSQL.
- Каждая реплицируемая таблица должна иметь один из следующих **репликационных идентификаторов**:
  - первичный ключ (по умолчанию)
  - индекс

```
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;
```

Первичный ключ всегда проверяется первым. Если он отсутствует, то проверяется индекс, определенный как `replica identity index` (репликационный идентификатор).

Если индекс используется в качестве репликационного идентификатора, то в таблице должен быть только один такой индекс.

Вы можете проверить, какой тип используется для указанной таблицы, выполнив следующую команду:

```
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;
```

## Предупреждение

Репликация **TOAST**-значений не поддерживается. Для типа данных будет использоваться значение по умолчанию.

## Пример использования

```
CREATE DATABASE postgresql_db
ENGINE = MaterializedPostgreSQL('postgres1:5432', 'postgres_database', 'postgres_user', 'postgres_password');

SELECT * FROM postgresql_db.postgres_table;
```

## Примечания

Сбой слота логической репликации

Слоты логической репликации, которые есть на основном сервере, не доступны на резервных репликах.

Поэтому в случае сбоя новый основной сервер (который раньше был резервным) не будет знать о слотах репликации, которые были созданы на вышедшем из строя основном сервере. Это приведет к нарушению репликации из PostgreSQL.

Решением этой проблемы может стать ручное управление слотами репликации и определение постоянного слота репликации (об этом можно прочитать [здесь](#)). Этот слот нужно передать с помощью настройки `materialized_postgresql_replication_slot`, и он должен быть экспортирован в параметре `EXPORT SNAPSHOT`. Идентификатор снэпшота нужно передать в настройке `materialized_postgresql_snapshot`.

Имейте в виду, что это стоит делать только если есть реальная необходимость. Если такой необходимости нет, или если нет полного понимания того, как это работает, то самостоятельно слот репликации конфигурировать не стоит, он будет создан таблицей.

### Пример (от [@bchrobot](#))

1. Сконфигурируйте слот репликации в PostgreSQL.

```
apiVersion: "acid.zalan.do/v1"
kind: postgresql
metadata:
  name: acid-demo-cluster
spec:
  numberOflInstances: 2
  postgresql:
    parameters:
      wal_level: logical
  patroni:
    slots:
      clickhouse_sync:
        type: logical
        database: demodb
        plugin: pgoutput
```

2. Дождитесь готовности слота репликации, затем инициируйте транзакцию и экспортируйте идентификатор снэпшота этой транзакции:

```
BEGIN;
SELECT pg_export_snapshot();
```

3. Создайте базу данных в ClickHouse:

```
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. Когда начнет выполняться репликация БД в ClickHouse, прервите транзакцию в PostgreSQL. Убедитесь, что репликация продолжается после сбоя:

```
kubectl exec acid-demo-cluster-0 -c postgres -- su postgres -c 'patronictl failover --candidate acid-demo-cluster-1 --force'
```

Позволяет подключаться к базам данных на удалённом MySQL сервере и выполнять запросы `INSERT` и `SELECT` для обмена данными между ClickHouse и MySQL.

Движок баз данных MySQL транслирует запросы при передаче на сервер MySQL, что позволяет выполнять и другие виды запросов, например `SHOW TABLES` или `SHOW CREATE TABLE`.

Не поддерживаемые виды запросов:

- `RENAME`
- `CREATE TABLE`
- `ALTER`

## Создание базы данных

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

### Параметры движка

- `host:port` — адрес сервера MySQL.
- `database` — имя базы данных на удалённом сервере.
- `user` — пользователь MySQL.
- `password` — пароль пользователя.

## Поддержка типов данных

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

Все прочие типы данных преобразуются в [String](#).

[Nullable](#) поддержан.

## Использование глобальных переменных

Для лучшей совместимости к глобальным переменным можно обращаться в формате MySQL, как `@@identifier`.

Поддерживаются следующие переменные:

- `version`
- `max_allowed_packet`

### Предупреждение

В настоящее время эти переменные реализованы только как "заглушки" и не содержат актуальных данных.

Пример:

```
SELECT @@version;
```

## Примеры использования

Таблица в 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)
```

База данных в ClickHouse, позволяющая обмениваться данными с сервером MySQL:

```
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

Сохраняет таблицы только в оперативной памяти `expiration_time_in_seconds` через несколько секунд после последнего доступа. Может использоваться только с таблицами \*Log.

Он оптимизирован для хранения множества небольших таблиц \*Log, для которых обычно существует большой временной интервал между обращениями.

## Создание базы данных

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

## Atomic

Поддерживает неблокирующие запросы `DROP TABLE` и `RENAME TABLE` и атомарные запросы `EXCHANGE TABLES`. Двигок Atomic используется по умолчанию.

## Создание БД

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

# Особенности и рекомендации

## UUID

Каждая таблица в базе данных Atomic имеет уникальный **UUID** и хранит данные в папке `/clickhouse_path/store/xxx/xxxxyyyy-yyyy-yyyy-yyyyyyyyyy/`, где `xxxxyyyy-yyyy-yyyy-yyyyyyyyyy` - это UUID таблицы.

Обычно UUID генерируется автоматически, но пользователь также может явно указать UUID в момент создания таблицы (однако это не рекомендуется). Для отображения UUID в запросе `SHOW CREATE` вы можете использовать настройку `show_table_uuid_in_table_create_query_if_not_nil`. Результат выполнения в таком случае будет иметь вид:

```
CREATE TABLE name UUID '28f1c61c-2970-457a-bffe-454156ddcfef' (n UInt64) ENGINE = ...;
```

## RENAME TABLE

Запросы **RENAME** выполняются без изменения UUID и перемещения табличных данных. Эти запросы не ожидают завершения использующих таблицу запросов и выполняются мгновенно.

## DROP/DETACH TABLE

При выполнении запроса `DROP TABLE` никакие данные не удаляются. Таблица помечается как удаленная, метаданные перемещаются в папку `/clickhouse_path/metadata_dropped/` и база данных уведомляет фоновый поток. Задержка перед окончательным удалением данных задается настройкой `database_atomic_delay_before_drop_table_sec`.

Вы можете задать синхронный режим, определяя модификатор `SYNC`. Используйте для этого настройку `database_atomic_wait_for_drop_and_detach_synchronously`. В этом случае запрос `DROP` ждет завершения `SELECT`, `INSERT` и других запросов, которые используют таблицу. Таблица будет фактически удалена, когда она не будет использоваться.

## EXCHANGE TABLES/DICTIONARIES

Запрос **EXCHANGE** атомарно меняет местами две таблицы или два словаря. Например, вместо неатомарной операции:

```
RENAME TABLE new_table TO tmp, old_table TO new_table, tmp TO old_table;
```

вы можете использовать один атомарный запрос:

```
EXCHANGE TABLES new_table AND old_table;
```

## ReplicatedMergeTree in Atomic Database

Для таблиц **ReplicatedMergeTree** рекомендуется не указывать параметры движка - путь в ZooKeeper и имя реплики. В этом случае будут использоваться параметры конфигурации: `default_replica_path` и `default_replica_name`. Если вы хотите определить параметры движка явно, рекомендуется использовать макрос `{uuid}`. Это удобно, так как автоматически генерируются уникальные пути для каждой таблицы в ZooKeeper.

## Смотрите также

- Системная таблица `system.databases`.

## SQLite

Движок баз данных позволяет подключаться к базе **SQLite** и выполнять запросы **INSERT** и **SELECT** для обмена данными между ClickHouse и SQLite.

## Создание базы данных

```
CREATE DATABASE sqlite_database  
ENGINE = SQLite('db_path')
```

### Параметры движка

- `db_path` — путь к файлу с базой данных SQLite.

## Поддерживаемые типы данных

SQLite	ClickHouse
INTEGER	Int32
REAL	Float32
TEXT	String
BLOB	String

## Особенности и рекомендации

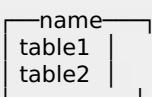
SQLite хранит всю базу данных (определения, таблицы, индексы и сами данные) в виде единого кроссплатформенного файла на хост-машине. Во время записи SQLite блокирует весь файл базы данных, поэтому операции записи выполняются последовательно. Операции чтения могут быть многозадачными.

SQLite не требует управления службами (например, сценариями запуска) или контроля доступа на основе `GRANT` и паролей. Контроль доступа осуществляется с помощью разрешений файловой системы, предоставляемых самому файлу базы данных.

## Примеры использования

Отобразим список таблиц базы данных в ClickHouse, подключенной к SQLite:

```
CREATE DATABASE sqlite_db ENGINE = SQLite('sqlite.db');  
SHOW TABLES FROM sqlite_db;
```



Отобразим содержимое таблицы:

```
SELECT * FROM sqlite_db.table1;
```

col1	col2
line1	1
line2	2
line3	3

Вставим данные в таблицу SQLite из таблицы ClickHouse:

```
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

Позволяет подключаться к БД на удаленном сервере PostgreSQL. Поддерживает операции чтения и записи (запросы SELECT и INSERT) для обмена данными между ClickHouse и PostgreSQL.

Позволяет в реальном времени получать от удаленного сервера PostgreSQL информацию о таблицах БД и их структуре с помощью запросов SHOW TABLES и DESCRIBE TABLE.

Поддерживает операции изменения структуры таблиц (ALTER TABLE ... ADD|DROP COLUMN). Если параметр use\_table\_cache (см. ниже раздел Параметры движка) установлен в значение 1, структура таблицы кешируется, и изменения в структуре не отслеживаются, но будут обновлены, если выполнить команды DETACH и ATTACH.

## Создание БД

```
CREATE DATABASE test_database
ENGINE = PostgreSQL('host:port', 'database', 'user', 'password'[, `schema`, `use_table_cache`]);
```

### Параметры движка

- host:port — адрес сервера PostgreSQL.
- database — имя удаленной БД.
- user — пользователь PostgreSQL.
- password — пароль пользователя.
- schema — схема PostgreSQL.
- use\_table\_cache — определяет кеширование структуры таблиц БД. Необязательный параметр. Значение по умолчанию: 0.

## Поддерживаемые типы данных

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

## Примеры использования

Обмен данными между Бд ClickHouse и сервером PostgreSQL:

```
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	

Чтение данных из таблицы PostgreSQL:

```
SELECT * FROM test_database.test_table;
```

id	value
1	2

Запись данных в таблицу PostgreSQL:

```
INSERT INTO test_database.test_table VALUES (3,4);
SELECT * FROM test_database.test_table;
```

int_id	value
1	2
3	4

Пусть структура таблицы была изменена в PostgreSQL:

```
postgres> ALTER TABLE test_table ADD COLUMN data Text
```

Поскольку при создании БД параметр `use_table_cache` был установлен в значение 1, структура таблицы в ClickHouse была кеширована и поэтому не изменилась:

```
DESCRIBE TABLE test_database.test_table;
```

name	type
id	Nullable(Integer)
value	Nullable(Integer)

После того как таблицу «отцепили» и затем снова «прицепили», структура обновилась:

```
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)

## [экспериментальный] Replicated

Движок основан на движке **Atomic**. Он поддерживает репликацию метаданных через журнал DDL, записываемый в ZooKeeper и выполняемый на всех репликах для данной базы данных.

На одном сервере ClickHouse может одновременно работать и обновляться несколько реплицированных баз данных. Но не может существовать нескольких реплик одной и той же реплицированной базы данных.

## Создание базы данных

```
CREATE DATABASE testdb ENGINE = Replicated('zoo_path', 'shard_name', 'replica_name') [SETTINGS ...]
```

## Параметры движка

- `zoo_path` — путь в ZooKeeper. Один и тот же путь ZooKeeper соответствует одной и той же базе данных.
- `shard_name` — Имя шарда. Реплики базы данных группируются в шарды по имени.
- `replica_name` — Имя реплики. Имена реплик должны быть разными для всех реплик одного и того же шарда.

## Предупреждение

Для таблиц **ReplicatedMergeTree** если аргументы не заданы, то используются аргументы по умолчанию: `/clickhouse/tables/{uuid}/{shard}` и `{replica}`. Они могут быть изменены в серверных настройках: `default_replica_path` и `default_replica_name`. Макрос `{uuid}` раскрывается в UUID таблицы, `{shard}` и `{replica}` — в значения из конфига сервера. В будущем появится возможность использовать значения `shard_name` и `replica_name` аргументов движка базы данных **Replicated**.

## Особенности и рекомендации

DDL-запросы с базой данных **Replicated** работают похожим образом на **ON CLUSTER** запросы, но с небольшими отличиями.

Сначала DDL-запрос пытается выполниться на инициаторе (том хосте, который изначально получил запрос от пользователя). Если запрос не выполнился, то пользователь сразу получает ошибку, другие хосты не пытаются его выполнить. Если запрос успешно выполнился на инициаторе, то все остальные хосты будут автоматически делать попытки выполнить его.

Инициатор попытается дождаться выполнения запроса на других хостах (не дольше `distributed_ddl_task_timeout`) и вернёт таблицу со статусами выполнения запроса на каждом хосте.

Поведение в случае ошибок регулируется настройкой `distributed_ddl_output_mode`, для **Replicated** лучше выставлять её в `null_status_on_timeout` — т.е. если какие-то хосты не успели выполнить запрос за `distributed_ddl_task_timeout`, то вместо исключения для них будет показан статус `NULL` в таблице.

В системной таблице `system.clusters` есть кластер с именем, как у реплицируемой базы, который состоит из всех реплик базы. Этот кластер обновляется автоматически при создании/удалении реплик, и его можно использовать для **Distributed** таблиц.

При создании новой реплики базы, эта реплика сама создаёт таблицы. Если реплика долго была недоступна и отстала от лога репликации — она сверяет свои локальные метаданные с актуальными метаданными в ZooKeeper, перекладывает лишние таблицы с данными в отдельную нереплицируемую базу (чтобы случайно не удалить что-нибудь лишнее), создаёт недостающие таблицы, обновляет имена таблиц, если были переименования. Данные реплицируются на уровне **ReplicatedMergeTree**, т.е. если таблица не реплицируемая, то данные реплицироваться не будут (база отвечает только за метаданные).

## Примеры использования

Создадим реплицируемую базу на трех хостах:

```
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}');
```

Выполним DDL-запрос на одном из хостов:

```
CREATE TABLE r.rmt (n UInt64) ENGINE=ReplicatedMergeTree ORDER BY n;
```

Запрос выполнится на всех остальных хостах:

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	

Кластер в системной таблице system.clusters:

```
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

Создадим распределенную таблицу и вставим в нее данные:

```
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]

Добавление реплики:

```
node4 :) CREATE DATABASE r ENGINE=Replicated('some/path/r','other_shard','r2');
```

Новая реплика автоматически создаст все таблицы, которые есть в базе, а старые реплики перезагрусят из ZooKeeper-а конфигурацию кластера:

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

Распределенная таблица также получит данные от нового хоста:

```
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]

## Справка по SQL

- [SELECT](#)
- [INSERT INTO](#)
- [CREATE](#)
- [ALTER](#)
- [Прочие виды запросов](#)

## SQL выражения в ClickHouse

Выражения описывают различные действия, которые можно выполнить с помощью SQL запросов. Каждый вид выражения имеет свой синтаксис и особенности использования, которые описаны в соответствующих разделах документации:

- [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

## Синтаксис запросов SELECT

SELECT выполняет получение данных.

```
[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 ALL ...]
[INTO OUTFILE filename]
[FORMAT format]
```

Все секции являются необязательными, за исключением списка выражений сразу после SELECT, о котором более подробно будет рассказано [ниже](#).

Особенности каждой необязательной секции рассматриваются в отдельных разделах, которые перечислены в том же порядке, в каком они выполняются:

- Секция WITH
- Секция SELECT
- Секция DISTINCT
- Секция FROM
- Секция SAMPLE
- Секция JOIN
- Секция PREWHERE
- Секция WHERE
- Секция GROUP BY
- Секция LIMIT BY
- Секция HAVING
- Секция LIMIT
- Секция OFFSET
- Секция UNION ALL
- Секция INTERSECT
- Секция EXCEPT

- Секция INTO OUTFILE
- Секция FORMAT

## Секция SELECT

Выражения указанные в секции `SELECT` анализируются после завершения всех вычислений из секций, описанных выше. Вернее, анализируются выражения, стоящие над агрегатными функциями, если есть агрегатные функции.

Сами агрегатные функции и то, что под ними, вычисляются при агрегации (`GROUP BY`). Эти выражения работают так, как будто применяются к отдельным строкам результата.

Если в результат необходимо включить все столбцы, используйте символ звёздочки (\*). Например, `SELECT * FROM ...`

Чтобы включить в результат несколько столбцов, выбрав их имена с помощью регулярных выражений `re2`, используйте выражение `COLUMNS`.

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

Например, рассмотрим таблицу:

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

Следующий запрос выбирает данные из всех столбцов, содержащих в имени символ a.

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

aa	ab
1	1

Выбранные столбцы возвращаются не в алфавитном порядке.

В запросе можно использовать несколько выражений `COLUMNS`, а также вызывать над ними функции.

Например:

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

aa	ab	bc	toTypeName(bc)
1	1	1	Int8

Каждый столбец, возвращённый выражением `COLUMNS`, передаётся в функцию отдельным аргументом. Также можно передавать и другие аргументы, если функция их поддерживает. Аккуратно используйте функции. Если функция не поддерживает переданное количество аргументов, то ClickHouse генерирует исключение.

Например:

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

```
Received exception from server (version 19.14.1):
Code: 42. DB::Exception: Received from localhost:9000. DB::Exception: Number of arguments for function plus
doesn't match: passed 3, should be 2.
```

В этом примере, `COLUMNS('a')` возвращает два столбца: `aa` и `ab`. `COLUMNS('c')` возвращает столбец `bc`. Оператор `+` не работает с тремя аргументами, поэтому ClickHouse генерирует исключение с соответствующим сообщением.

Столбцы, которые возвращаются выражением `COLUMNS` могут быть разных типов. Если `COLUMNS` не возвращает ни одного столбца и это единственное выражение в запросе `SELECT`, то ClickHouse генерирует исключение.

## Звёздочка

В любом месте запроса, вместо выражения, может стоять звёздочка. При анализе запроса звёздочка раскрывается в список всех столбцов таблицы (за исключением `MATERIALIZED` и `ALIAS` столбцов). Есть лишь немногих случаев, когда оправдано использовать звёздочку:

- при создании дампа таблицы;
- для таблиц, содержащих всего несколько столбцов - например, системных таблиц;
- для получения информации о том, какие столбцы есть в таблице; в этом случае, укажите `LIMIT 1`. Но лучше используйте запрос `DESC TABLE`;
- при наличии сильной фильтрации по небольшому количеству столбцов с помощью `PREWHERE`;
- в подзапросах (так как из подзапросов выкидываются столбцы, не нужные для внешнего запроса).

В других случаях использование звёздочки является издевательством над системой, так как вместо преимуществ столбцовой СУБД вы получаете недостатки. То есть использовать звёздочку не рекомендуется.

## Экстремальные значения

Вы можете получить в дополнение к результату также минимальные и максимальные значения по столбцам результата. Для этого выставьте настройку `extremes` в 1. Минимумы и максимумы считаются для числовых типов, дат, дат-с-временем. Для остальных столбцов будут выведены значения по умолчанию.

Вычисляются дополнительные две строчки - минимумы и максимумы, соответственно. Эти две дополнительные строки выводятся в `форматах JSON*`, `TabSeparated*`, и `Pretty*` отдельно от остальных строчек. В остальных форматах они не выводятся.

Во форматах `JSON*`, экстремальные значения выводятся отдельным полем 'extremes'. В форматах `TabSeparated*`, строка выводится после основного результата и после 'totals' если есть. Перед ней (после остальных данных) вставляется пустая строка. В форматах `Pretty*`, строка выводится отдельной таблицей после основного результата и после `totals` если есть.

Экстремальные значения вычисляются для строк перед `LIMIT`, но после `LIMIT BY`. Однако при использовании `LIMIT offset, size`, строки перед `offset` включаются в `extremes`. В потоковых запросах, в результате может учитываться также небольшое количество строчек, прошедших `LIMIT`.

## Замечания

Вы можете использовать синонимы (алиасы `AS`) в любом месте запроса.

В секциях `GROUP BY`, `ORDER BY` и `LIMIT BY` можно использовать не названия столбцов, а номера. Для этого нужно включить настройку `enable_positional_arguments`. Тогда, например, в запросе с `ORDER BY 1,2` будет выполнена сортировка сначала по первому, а затем по второму столбцу.

## Детали реализации

Если в запросе отсутствуют секции `DISTINCT`, `GROUP BY`, `ORDER BY`, подзапросы в `IN` и `JOIN`, то запрос будет обработан полностью потоково, с использованием  $O(1)$  количества оперативки.

Иначе запрос может съесть много оперативки, если не указаны подходящие ограничения:

- `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`

Подробнее смотрите в разделе «Настройки». Присутствует возможность использовать внешнюю сортировку (с сохранением временных данных на диск) и внешнюю агрегацию.

## Модификаторы запроса `SELECT`

Вы можете использовать следующие модификаторы в запросах `SELECT`.

### APPLY

Вызывает указанную функцию для каждой строки, возвращаемой внешним табличным выражением запроса.

#### Синтаксис:

```
SELECT <expr> APPLY( <func> ) FROM [db.]table_name
```

#### Пример:

```
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

Исключает из результата запроса один или несколько столбцов.

#### Синтаксис:

```
SELECT <expr> EXCEPT ( col_name1 [, col_name2, col_name3, ...] ) FROM [db.]table_name
```

#### Пример:

```
SELECT * EXCEPT (i) from columns_transformers;
```

j	k
10	324
8	23

## REPLACE

Определяет одно или несколько выражений алиасов. Каждый алиас должен соответствовать имени столбца из запроса SELECT \*. В списке столбцов результата запроса имя столбца, соответствующее алиасу, заменяется выражением в модификаторе REPLACE.

Этот модификатор не изменяет имена или порядок столбцов. Однако он может изменить значение и тип значения.

#### Синтаксис:

```
SELECT <expr> REPLACE( <expr> AS col_name) from [db.]table_name
```

#### Пример:

```
SELECT * REPLACE(i + 1 AS i) from columns_transformers;
```

i	j	k
101	10	324
121	8	23

## Комбинации модификаторов

Вы можете использовать каждый модификатор отдельно или комбинировать их.

#### Примеры:

Использование одного и того же модификатора несколько раз.

```
SELECT COLUMNS('jk') APPLY(toString) APPLY(length) APPLY(max) from columns_transformers;
```

max(length(toString(j)))	max(length(toString(k)))
2	3

Использование нескольких модификаторов в одном запросе.

```
SELECT * REPLACE(i + 1 AS i) EXCEPT (j) APPLY(sum) from columns_transformers;
```

sum(plus(i, 1))	sum(k)
222	347

## SETTINGS в запросе SELECT

Вы можете задать значения необходимых настроек непосредственно в запросе `SELECT` в секции `SETTINGS`. Эти настройки действуют только в рамках данного запроса, а после его выполнения сбрасываются до предыдущего значения или значения по умолчанию.

Другие способы задания настроек описаны [здесь](#).

### Пример

```
SELECT * FROM some_table SETTINGS optimize_read_in_order=1, cast_keep_nullable=1;
```

## Секция ALL

Если в таблице несколько совпадающих строк, то `ALL` возвращает все из них. Поведение запроса `SELECT ALL` точно такое же, как и `SELECT` без аргумента `DISTINCT`. Если указаны оба аргумента: `ALL` и `DISTINCT`, функция вернет исключение.

`ALL` может быть указан внутри агрегатной функции, например, результат выполнения запроса:

```
SELECT sum(ALL number) FROM numbers(10);
```

равен результату выполнения запроса:

```
SELECT sum(number) FROM numbers(10);
```

## Секция ARRAY JOIN

Типовая операция для таблиц, содержащих столбец-массив — произвести новую таблицу, которая будет иметь столбец с каждым отдельным элементом массивов из изначального столбца, в то время как значения других столбцов дублируются. Это основной сценарий использования секции `ARRAY JOIN`.

Название этой секции происходит от того, что эту операцию можно рассматривать как исполняющий `JOIN` с массивом или вложенной структурой данных. Цель использования похожа на функцию `arrayJoin`, но функциональность секции шире.

Синтаксис:

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

Вы можете указать только одну секцию `ARRAY JOIN` в `SELECT` запросе.

Поддерживаемые виды ARRAY JOIN перечислены ниже:

- **ARRAY JOIN** - В базовом случае пустые массивы не включаются в результат.
- **LEFT ARRAY JOIN** - Результат содержит строки с пустыми массивами. Значение для пустого массива устанавливается равным значению по умолчанию для типа элемента массива (обычно 0, пустая строка или NULL).

## Базовые примеры ARRAY JOIN

Приведенные ниже примеры демонстрируют использование **ARRAY JOIN** и **LEFT ARRAY JOIN**. Создадим таблицу с колонкой типа данных **Array** и вставим в него значения:

```
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	[]

В приведенном ниже примере используется секция **ARRAY JOIN**:

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

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

В следующем примере используется **LEFT ARRAY JOIN**:

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

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

## Использование алиасов

В секции `ARRAY JOIN` может быть указан алиас для массива. В этом случае элемент массива доступен по этому алиасу, а сам массив доступен по исходному имени. Пример:

```
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

Используя псевдонимы, вы можете выполнять `ARRAY JOIN` с внешними массивами. Например:

```
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

Несколько массивов могут быть перечислены через запятую в секции `ARRAY JOIN`. В этом случае, `JOIN` выполняется с ними одновременно (прямая сумма, а не декартово произведение). Обратите внимание, что все массивы должны иметь одинаковый размер. Пример:

```
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

В приведенном ниже примере используется функция `arrayEnumerate`:

```
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 со вложенной структурой данных

ARRAY JOIN также работает с **вложенными структурами данных**:

```
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

При указании имен вложенных структур данных в ARRAY JOIN, секция работает так же, как и ARRAY JOIN со всеми элементами массива, из которых он состоит. Примеры приведены ниже:

```
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

Такая вариация также возможна:

```
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]

Алиас для вложенной структуры данных можно использовать, чтобы выбрать как результат JOIN-а, так и исходный массив. Пример:

```
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]

Пример использования функции `arrayEnumerate`:

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

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

## Детали реализации

Хотя секция `ARRAY JOIN` всегда должна быть указана перед `WHERE/PREWHERE`, технически они могут быть выполнены в любом порядке, если только результат `ARRAY JOIN` используется для фильтрации. Порядок обработки контролируется оптимизатором запросов.

## Секция `DISTINCT`

Если указан `SELECT DISTINCT`, то в результате запроса останутся только уникальные строки. Таким образом, из всех наборов полностью совпадающих строк в результате останется только одна строка.

Вы можете указать столбцы, по которым хотите отбирать уникальные значения: `SELECT DISTINCT ON (column1, column2,...)`. Если столбцы не указаны, то отбираются строки, в которых значения уникальны во всех столбцах.

Рассмотрим таблицу:

a	b	c
1	1	1
1	1	1
2	2	2
2	2	2
1	1	2
1	2	2

Использование DISTINCT без указания столбцов:

```
SELECT DISTINCT * FROM t1;
```

a	b	c
1	1	1
2	2	2
1	1	2
1	2	2

Использование DISTINCT с указанием столбцов:

```
SELECT DISTINCT ON (a,b) * FROM t1;
```

a	b	c
1	1	1
2	2	2
1	2	2

## DISTINCT и ORDER BY

ClickHouse поддерживает использование секций DISTINCT и ORDER BY для разных столбцов в одном запросе. Секция DISTINCT выполняется до секции ORDER BY.

Таблица для примера:

a	b
2	1
1	2
3	3
2	4

При выборе данных с помощью `SELECT DISTINCT a FROM t1 ORDER BY b ASC` мы получаем следующий результат:

a
2
1
3

Если мы изменим направление сортировки `SELECT DISTINCT a FROM t1 ORDER BY b DESC` мы получаем следующий результат:

a
3
1
2

Ряд 2, 4 был разрезан перед сортировкой.

Учитывайте эту специфику при разработке запросов.

## Обработка NULL

`DISTINCT` работает с `NULL` как-будто `NULL` — обычное значение и `NULL==NULL`. Другими словами, в результате `DISTINCT`, различные комбинации с `NULL` встречаются только один раз. Это отличается от обработки `NULL` в большинстве других контекстов.

## Альтернативы

Можно получить такой же результат, применив `GROUP BY` для того же набора значений, которые указан в секции `SELECT`, без использования каких-либо агрегатных функций. Но есть несколько отличий от `GROUP BY`:

- `DISTINCT` может применяться вместе с `GROUP BY`.
- Когда секция `ORDER BY` опущена, а секция `LIMIT` присутствует, запрос прекращает выполнение сразу после считывания необходимого количества различных строк.
- Блоки данных выводятся по мере их обработки, не дожидаясь завершения выполнения всего запроса.

## Секция EXCEPT

`EXCEPT` возвращает только те строки, которые являются результатом первого запроса без результатов второго. В запросах количество, порядок следования и типы столбцов должны совпадать. Результат `EXCEPT` может содержать повторяющиеся строки.

Если используется несколько `EXCEPT`, и в выражении не указаны скобки, `EXCEPT` выполняется по порядку слева направо. `EXCEPT` имеет такой же приоритет выполнения, как `UNION`, и приоритет ниже, чем у `INTERSECT`.

```
SELECT column1 [, column2 ]
FROM table1
[WHERE condition]

EXCEPT

SELECT column1 [, column2 ]
FROM table2
[WHERE condition]
```

Условие в секции `WHERE` может быть любым в зависимости от ваших требований.

## Примеры

Запрос:

```
SELECT number FROM numbers(1,10) EXCEPT SELECT number FROM numbers(3,6);
```

Результат:

number
1
2
9
10

Запрос:

```
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;
```

Результат:

one	two	three
l	p	o
k	t	d
l	p	o

## См. также

- [UNION](#)
- [INTERSECT](#)

## Секция FORMAT

ClickHouse поддерживает широкий спектр [форматов сериализации](#) это может быть использовано, в частности, для результатов запросов. Существует несколько способов выбора формата для `SELECT`, один из них заключается в том, чтобы указать `FORMAT format` в конце запроса, чтобы получить результирующие данные в любом конкретном формате.

Определенный формат может использоваться для удобства, интеграции с другими системами или для повышения производительности.

## Формат по умолчанию

Если `FORMAT` предложение опущено, используется формат по умолчанию, который зависит как от настроек, так и от интерфейса, используемого для доступа к серверу ClickHouse. Для [HTTP-интерфейса и клиента командной строки](#) в пакетном режиме, формат по умолчанию — `TabSeparated`. Для клиента командной строки в интерактивном режиме по умолчанию используется формат `PrettyCompact` (он производит компактные человеческочитаемые таблицы).

## Детали реализации

При использовании клиента командной строки данные всегда передаются по сети во внутреннем эффективном формате (`Native`). Клиент самостоятельно интерпретирует `FORMAT` предложение запроса и форматирует сами данные (тем самым освобождая сеть и сервер от дополнительной нагрузки).

# Секция FROM

В секции `FROM` указывается источник, из которого будут читаться данные:

- Таблица
- Подзапрос
- Табличная функция

Секция `JOIN` и `ARRAY JOIN` могут быть использованы для расширения функциональных возможностей секции `FROM`.

Подзапрос — дополнительный `SELECT` запрос, который может быть указан в круглых скобках внутри секции `FROM`.

Секция `FROM` может содержать несколько источников данных, указанных через запятую, что эквивалентно выполнению `CROSS JOIN` на них.

## Модификатор FINAL

Если в запросе используется модификатор `FINAL`, то ClickHouse полностью мёржит данные перед выдачей результата, таким образом выполняя все преобразования данных, которые производятся движком таблиц при мёржах.

Он применим при выборе данных из таблиц, использующих `MergeTree`- семейство движков. Также поддерживается для:

- `Replicated` варианты исполнения `MergeTree` движков.
- `View`, `Buffer`, `Distributed`, и `MaterializedView`, которые работают поверх других движков, если они созданы для таблиц с движками семейства `MergeTree`.

Теперь `SELECT` запросы с `FINAL` выполняются параллельно и, следовательно, немного быстрее. Но имеются серьезные недостатки при их использовании (смотрите ниже). Настройка `max_final_threads` устанавливает максимальное количество потоков.

## Недостатки

Запросы, которые используют `FINAL` выполняются немного медленнее, чем аналогичные запросы без него, потому что:

- Данные мёржатся во время выполнения запроса.
- Запросы с модификатором `FINAL` читают столбцы первичного ключа в дополнение к столбцам, используемым в запросе.

**В большинстве случаев избегайте использования `FINAL`.** Общий подход заключается в использовании агрегирующих запросов, которые предполагают, что фоновые процессы движков семейства `MergeTree` ещё не случились (например, сами отбрасывают дубликаты).

## Детали реализации

Если секция `FROM` опущена, данные будут считываться из таблицы `system.one`.

Таблица `system.one` содержит ровно одну строку.

Для выполнения запроса, из соответствующей таблицы, вынимаются все столбцы, перечисленные в запросе. Из подзапросов выкидываются столбцы, не нужные для внешнего запроса.

Если в запросе не перечислено ни одного столбца (например, `SELECT count() FROM t`), то из таблицы всё равно вынимается один какой-нибудь столбец (предпочитается самый маленький), для того, чтобы можно было посчитать количество строк.

## Секция GROUP BY

Секция `GROUP BY` переключает `SELECT` запрос в режим агрегации, который работает следующим образом:

- Секция `GROUP BY` содержит список выражений (или одно выражение, которое считается списком длины один). Этот список действует как «ключ группировки», в то время как каждое отдельное выражение будет называться «ключевым выражением».
- Все выражения в секциях `SELECT`, `HAVING`, и `ORDER BY` статьи **должны** быть вычисляемыми на основе ключевых выражений **или** на **агрегатных функций** над неключевыми выражениями (включая столбцы). Другими словами, каждый столбец, выбранный из таблицы, должен использоваться либо в ключевом выражении, либо внутри агрегатной функции, но не в обоих.
- В результате агрегирования `SELECT` запрос будет содержать столько строк, сколько было уникальных значений ключа группировки в исходной таблице. Обычно агрегация значительно уменьшает количество строк, часто на порядки, но не обязательно: количество строк остается неизменным, если все исходные значения ключа группировки ценности были различны.

Если вы хотите для группировки данных в таблице указывать номера столбцов, а не названия, включите настройку `enable_positional_arguments`.

### Примечание

Есть ещё один способ запустить агрегацию по таблице. Если запрос содержит столбцы исходной таблицы только внутри агрегатных функций, то `GROUP BY` секцию можно опустить, и предполагается агрегирование по пустому набору ключей. Такие запросы всегда возвращают ровно одну строку.

## Обработка NULL

При агрегации ClickHouse интерпретирует `NULL` как обычное значение, то есть `NULL==NULL`. Это отличается от обработки `NULL` в большинстве других контекстов.

Предположим, что у вас есть эта таблица:

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

Запрос `SELECT sum(x), y FROM t_null_big GROUP BY y` выведет:

sum(x)	y
4	2
3	3
5	NULL

Видно, что `GROUP BY` для `Y = NULL` просуммировал `x`, как будто `NULL` — это значение.

Если в `GROUP BY` передать несколько ключей, то в результате мы получим все комбинации выборки, как если бы `NULL` был конкретным значением.

## Модификатор WITH ROLLUP

Модификатор `WITH ROLLUP` применяется для подсчета подытогов для ключевых выражений. При этом учитывается порядок следования ключевых выражений в списке `GROUP BY`. Подытоги подсчитываются в обратном порядке: сначала для последнего ключевого выражения в списке, потом для предпоследнего и так далее вплоть до самого первого ключевого выражения.

Строки с подытогами добавляются в конец результирующей таблицы. В колонках, по которым строки уже сгруппированы, указывается значение `0` или пустая строка.

### Примечание

Если в запросе есть секция `HAVING`, она может повлиять на результаты расчета подытогов.

### Пример

Рассмотрим таблицу `t`:

year	month	day
2019	1	5
2019	1	15
2020	1	5
2020	1	15
2020	10	5
2020	10	15

Запрос:

```
SELECT year, month, day, count(*) FROM t GROUP BY year, month, day WITH ROLLUP;
```

Поскольку секция `GROUP BY` содержит три ключевых выражения, результат состоит из четырех таблиц с подытогами, которые как бы "сворачиваются" справа налево:

- `GROUP BY year, month, day`;
- `GROUP BY year, month` (а колонка `day` заполнена нулями);
- `GROUP BY year` (теперь обе колонки `month, day` заполнены нулями);
- и общий итог (все три колонки с ключевыми выражениями заполнены нулями).

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

## Модификатор WITH CUBE

Модификатор `WITH CUBE` применяется для расчета подытогов по всем комбинациям группировки ключевых выражений в списке `GROUP BY`.

Строки с подытогами добавляются в конец результирующей таблицы. В колонках, по которым выполняется группировка, указывается значение 0 или пустая строка.

### Примечание

Если в запросе есть секция `HAVING`, она может повлиять на результаты расчета подытогов.

### Пример

Рассмотрим таблицу 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;
```

Поскольку секция `GROUP BY` содержит три ключевых выражения, результат состоит из восьми таблиц с подытогами — по таблице для каждой комбинации ключевых выражений:

- `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

- и общий итог.

Колонки, которые не участвуют в GROUP BY, заполнены нулями.

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

## Модификатор WITH TOTALS

Если указан модификатор `WITH TOTALS`, то будет посчитана ещё одна строчка, в которой в столбцах-ключах будут содержаться значения по умолчанию (нули, пустые строки), а в столбцах агрегатных функций - значения, посчитанные по всем строкам («тотальные» значения).

Этот дополнительный ряд выводится только в форматах `JSON*`, `TabSeparated*`, и `Pretty*`, отдельно от других строк:

- В `JSON*` форматах, эта строка выводится как отдельное поле 'totals'.
- В `TabSeparated*` форматах, строка идет после основного результата, через дополнительную пустую строку (после остальных данных).
- В `Pretty*` форматах, строка выводится в виде отдельной таблицы после основного результата.
- В других форматах она не доступна.

При использовании секции **HAVING** поведение **WITH TOTALS** контролируется настройкой `totals_mode`.

## Настройка обработки итогов

По умолчанию `totals_mode = 'before_having'`. В этом случае totals считается по всем строчкам, включая непрощедших через HAVING и `max_rows_to_group_by`.

Остальные варианты учитывают в totals только строчки, прошедшие через HAVING, и имеют разное поведение при наличии настройки `max_rows_to_group_by` и `group_by_overflow_mode = 'any'`.

`after_having_exclusive` - не учитывать строчки, не прошедшие `max_rows_to_group_by`. То есть в totals попадёт меньше или столько же строчек, чем если бы `max_rows_to_group_by` не было.

`after_having_inclusive` - учитывать в totals все строчки, не прошедшие `max_rows_to_group_by`. То есть в totals попадёт больше или столько же строчек, чем если бы `max_rows_to_group_by` не было.

`after_having_auto` - считать долю строчек, прошедших через HAVING. Если она больше некоторого значения (по умолчанию - 50%), то включить все строчки, не прошедшие `max_rows_to_group_by` в totals, иначе - не включить.

`totals_auto_threshold` - по умолчанию 0.5. Коэффициент для работы `after_having_auto`.

Если `max_rows_to_group_by` и `group_by_overflow_mode = 'any'` не используются, то все варианты вида `after_having` не отличаются, и вы можете использовать любой из них, например, `after_having_auto`.

Вы можете использовать **WITH TOTALS** в подзапросах, включая подзапросы в секции **JOIN** (в этом случае соответствующие тотальные значения будут соединены).

## Примеры

Пример:

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

В отличие от MySQL (и в соответствии со стандартом SQL), вы не можете получить какое-нибудь значение некоторого столбца, не входящего в ключ или агрегатную функцию (за исключением константных выражений). Для обхода этого вы можете воспользоваться агрегатной функцией `any` (получить первое попавшееся значение) или `min/max`.

Пример:

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

GROUP BY вычисляет для каждого встретившегося различного значения ключей, набор значений агрегатных функций.

## Детали реализации

Агрегация является одной из наиболее важных возможностей столбцовых СУБД, и поэтому её реализация является одной из наиболее сильно оптимизированных частей ClickHouse. По умолчанию агрегирование выполняется в памяти с помощью хэш-таблицы. Она имеет более 40 специализаций, которые выбираются автоматически в зависимости от типов данных ключа группировки.

## Оптимизация GROUP BY для отсортированных таблиц

Агрегирование данных в отсортированных таблицах может выполняться более эффективно, если выражение GROUP BY содержит хотя бы префикс ключа сортировки или инъективную функцию с этим ключом. В таких случаях в момент считывания из таблицы нового значения ключа сортировки промежуточный результат агрегирования будет финализироваться и отправляться на клиентскую машину. Чтобы включить такой способ выполнения запроса, используйте настройку `optimize_aggregation_in_order`. Подобная оптимизация позволяет сэкономить память во время агрегации, но в некоторых случаях может привести к увеличению времени выполнения запроса.

## Группировка во внешней памяти

Можно включить сброс временных данных на диск, чтобы ограничить потребление оперативной памяти при выполнении GROUP BY.

Настройка `max_bytes_before_external_group_by` определяет пороговое значение потребления RAM, по достижении которого временные данные GROUP BY сбрасываются в файловую систему. Если равно 0 (по умолчанию) - значит выключено.

При использовании `max_bytes_before_external_group_by`, рекомендуем выставить `max_memory_usage` приблизительно в два раза больше. Это следует сделать, потому что агрегация выполняется в две стадии: чтение и формирование промежуточных данных (1) и слияние промежуточных данных (2). Сброс данных на файловую систему может производиться только на стадии 1. Если сброса временных данных не было, то на стадии 2 может потребляться до такого же объёма памяти, как на стадии 1.

Например, если `max_memory_usage` было выставлено в 10000000000, и вы хотите использовать внешнюю агрегацию, то имеет смысл выставить `max_bytes_before_external_group_by` в 10000000000, а `max_memory_usage` в 20000000000. При срабатывании внешней агрегации (если был хотя бы один сброс временных данных в файловую систему) максимальное потребление оперативки будет лишь чуть-чуть больше `max_bytes_before_external_group_by`.

При распределённой обработке запроса внешняя агрегация производится на удалённых серверах. Для того чтобы на сервере-инициаторе запроса использовалось немного оперативки, нужно выставить настройку `distributed_aggregation_memory_efficient` в 1.

## Секция HAVING

Позволяет фильтровать результаты агрегации, полученные с помощью GROUP BY. Разница с WHERE в том, что WHERE выполняется перед агрегацией, в то время как HAVING выполняется после него.

Из секции HAVING можно ссылаться на результаты агрегации из секции SELECT по их алиасу. Также секция HAVING может фильтровать по результатам дополнительных агрегаторов, которые не возвращаются в результатах запроса.

## Ограничения

HAVING нельзя использовать, если агрегация не выполняется. Вместо этого можно использовать WHERE.

# Секция INTERSECT

`INTERSECT` возвращает строки, которые есть только в результатах первого и второго запросов. В запросах должны совпадать количество столбцов, их порядок и тип. Результат `INTERSECT` может содержать повторяющиеся строки.

Если используется несколько `INTERSECT` и скобки не указаны, пересечение выполняется слева направо. У `INTERSECT` более высокий приоритет выполнения, чем у `UNION` и `EXCEPT`.

```
SELECT column1 [, column2 ]
FROM table1
[WHERE condition]
```

`INTERSECT`

```
SELECT column1 [, column2 ]
FROM table2
[WHERE condition]
```

Условие может быть любым в зависимости от ваших требований.

## Примеры

Запрос:

```
SELECT number FROM numbers(1,10) INTERSECT SELECT number FROM numbers(3,6);
```

Результат:

number
3
4
5
6
7
8

Запрос:

```
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');
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;
```

Результат:

one	two	three
q	m	b
s	d	f
s	d	f
s	d	f

## См. также

- [UNION](#)

- EXCEPT

## Секция INTO OUTFILE

Чтобы перенаправить вывод `SELECT` запроса в указанный файл на стороне клиента, добавьте к нему секцию `INTO OUTFILE filename` (где `filename` — строковый литерал).

## Детали реализации

- Эта функция доступна только в следующих интерфейсах: [клиент командной строки](#) и [clickhouse-local](#). Таким образом, запрос, отправленный через [HTTP интерфейс](#) вернет ошибку.
- Запрос завершится ошибкой, если файл с тем же именем уже существует.
- По умолчанию используется [выходной формат TabSeparated](#) (как в пакетном режиме клиента командной строки).

## Секция JOIN

`JOIN` создаёт новую таблицу путем объединения столбцов из одной или нескольких таблиц с использованием общих для каждой из них значений. Это обычная операция в базах данных с поддержкой SQL, которая соответствует `join` из [реляционной алгебры](#). Частный случай соединения одной таблицы часто называют `self-join`.

### Синтаксис

```
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>) ...
```

Выражения из секции `ON` и столбцы из секции `USING` называются «ключами соединения». Если не указано иное, при присоединение создаётся [Декартово произведение](#) из строк с совпадающими значениями ключей соединения, что может привести к получению результатов с гораздо большим количеством строк, чем исходные таблицы.

## Поддерживаемые типы соединения

Все типы из стандартного [SQL JOIN](#) поддерживаются:

- `INNER JOIN`, возвращаются только совпадающие строки.
- `LEFT OUTER JOIN`, не совпадающие строки из левой таблицы возвращаются в дополнение к совпадающим строкам.
- `RIGHT OUTER JOIN`, не совпадающие строки из правой таблицы возвращаются в дополнение к совпадающим строкам.
- `FULL OUTER JOIN`, не совпадающие строки из обеих таблиц возвращаются в дополнение к совпадающим строкам.
- `CROSS JOIN`, производит декартово произведение таблиц целиком, ключи соединения не указываются.

Без указания типа `JOIN` подразумевается `INNER`. Ключевое слово `OUTER` можно опускать. Альтернативным синтаксисом для `CROSS JOIN` является ли указание нескольких таблиц, разделённых запятыми, в [секции FROM](#).

Дополнительные типы соединений, доступные в ClickHouse:

- `LEFT SEMI JOIN` и `RIGHT SEMI JOIN`, белый список по ключам соединения, не производит декартово произведение.
- `LEFT ANTI JOIN` и `RIGHT ANTI JOIN`, черный список по ключам соединения, не производит декартово произведение.
- `LEFT ANY JOIN`, `RIGHT ANY JOIN` и `INNER ANY JOIN`, Частично (для противоположных сторон `LEFT` и `RIGHT`) или полностью (для `INNER` и `FULL`) отключает декартово произведение для стандартных видов `JOIN`.
- `ASOF JOIN` и `LEFT ASOF JOIN`, Для соединения последовательностей по нечеткому совпадению. Использование `ASOF JOIN` описано ниже.

## Примечание

Если настройка `join_algorithm` установлена в значение `partial_merge`, то для `RIGHT JOIN` и `FULL JOIN` поддерживается только уровень строгости `ALL` (`SEMI`, `ANTI`, `ANY` и `ASOF` не поддерживаются).

## Настройки

Значение строгости по умолчанию может быть переопределено с помощью настройки `join_default_strictness`.

Поведение сервера ClickHouse для операций `ANY JOIN` зависит от параметра `any_join_distinct_right_table_keys`.

### См. также

- [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](#)

## Условия в секции ON

Секция `ON` может содержать несколько условий, связанных оператором `AND`. Условия, задающие ключи соединения, должны содержать столбцы левой и правой таблицы и должны использовать оператор равенства. Прочие условия могут использовать другие логические операторы, но в отдельном условии могут использоваться столбцы либо только левой, либо только правой таблицы. Строки объединяются только тогда, когда всё составное условие выполнено. Если оно не выполнено, то строки могут попасть в результат в зависимости от типа `JOIN`. Обратите внимание, что если то же самое условие поместить в секцию `WHERE`, то строки, для которых оно не выполняется, никогда не попадут в результат.

## Примечание

Оператор `OR` внутри секции `ON` пока не поддерживается.

## Примечание

Если в условии использованы столбцы из разных таблиц, то пока поддерживается только оператор равенства (=).

## Пример

Рассмотрим `table_1` и `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

Запрос с одним условием, задающим ключ соединения, и дополнительным условием для `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');
```

Обратите внимание, что результат содержит строку с именем C и пустым текстом. Стока включена в результат, потому что использован тип соединения OUTER.

name	text
A	Text A
B	Text B
C	

Запрос с типом соединения INNER и несколькими условиями:

```
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');
```

Результат:

name	text	scores
B	Text B	15

## Использование ASOF JOIN

ASOF JOIN применим в том случае, когда необходимо объединять записи, которые не имеют точного совпадения.

Для работы алгоритма необходим специальный столбец в таблицах. Этот столбец:

- Должен содержать упорядоченную последовательность.
- Может быть одного из следующих типов: `Int`, `UInt`, `Float`, `Date`, `DateTime`, `Decimal`.
- Не может быть единственным столбцом в секции `JOIN`.

Синтаксис ASOF JOIN ... ON:

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

Можно использовать произвольное количество условий равенства и одно условие на ближайшее совпадение. Например, `SELECT count() FROM table_1 ASOF LEFT JOIN table_2 ON table_1.a == table_2.b AND table_2.t <= table_1.t.`

Условия, поддержанные для проверки на ближайшее совпадение: `>`, `>=`, `<`, `<=`.

Синтаксис ASOF JOIN ... USING:

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

Для слияния по равенству ASOF JOIN использует `equi_columnX`, а для слияния по ближайшему совпадению использует `asof_column` с условием `table_1.asof_column >= table_2.asof_column`. Столбец `asof_column` должен быть последним в секции `USING`.

Например, рассмотрим следующие таблицы:

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 принимает метку времени пользовательского события из `table_1` и находит такое событие в `table_2` метка времени которого наиболее близка к метке времени события из `table_1` в соответствии с условием на ближайшее совпадение. При этом столбец `user_id` используется для объединения по равенству, а столбец `ev_time` для объединения по ближайшему совпадению. В нашем примере `event_1_1` может быть объединено с `event_2_1`, `event_1_2` может быть объединено с `event_2_3`, а `event_2_2` не объединяется.

## Примечание

ASOF JOIN не поддержан для движка таблиц **Join**.

Чтобы задать значение строгости по умолчанию, используйте сессионный параметр `join_default_strictness`.

## Распределённый JOIN

Есть два пути для выполнения соединения с участием распределённых таблиц:

- При использовании обычного `JOIN`, запрос отправляется на удалённые серверы. На каждом из них выполняются подзапросы для формирования «правой» таблицы, и с этой таблицей выполняется соединение. То есть, «правая» таблица формируется на каждом сервере отдельно.

- При использовании `GLOBAL ... JOIN`, сначала сервер-инициатор запускает подзапрос для вычисления правой таблицы. Эта временная таблица передаётся на каждый удалённый сервер, и на них выполняются запросы с использованием переданных временных данных.

Будьте аккуратны при использовании `GLOBAL`. За дополнительной информацией обращайтесь в раздел [Распределенные подзапросы](#).

## Неявные преобразования типов

Запросы `INNER JOIN`, `LEFT JOIN`, `RIGHT JOIN` и `FULL JOIN` поддерживают неявные преобразования типов для ключей соединения. Однако запрос не может быть выполнен, если не существует типа, к которому можно привести значения ключей с обеих сторон (например, нет типа, который бы одновременно вмещал в себя значения `UInt64` и `Int64`, или `String` и `Int32`).

### Пример

Рассмотрим таблицу `t_1`:

a	b	toTypeName(a)	toTypeName(b)
1	1	UInt16	UInt8
2	2	UInt16	UInt8

и таблицу `t_2`:

a	b	toTypeName(a)	toTypeName(b)
-1	1	Int16	Nullable(Int64)
1	-1	Int16	Nullable(Int64)
1	1	Int16	Nullable(Int64)

Запрос

```
SELECT a, b, toTypeName(a), toTypeName(b) FROM t_1 FULL JOIN t_2 USING (a, b);
```

вернёт результат:

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)

## Рекомендации по использованию

### Обработка пустых ячеек и NULL

При соединении таблиц могут появляться пустые ячейки. Настройка `join_use_nulls` определяет, как ClickHouse заполняет эти ячейки.

Если ключами `JOIN` выступают поля типа `Nullable`, то строки, где хотя бы один из ключей имеет значение `NULL`, не соединяются.

## Синтаксис

Требуется, чтобы столбцы, указанные в `USING`, назывались одинаково в обоих подзапросах, а остальные столбцы - по-разному. Изменить имена столбцов в подзапросах можно с помощью синонимов.

В секции `USING` указывается один или несколько столбцов для соединения, что обозначает условие на равенство этих столбцов. Список столбцов задаётся без скобок. Более сложные условия соединения не поддерживаются.

## Ограничения синтаксиса

Для множественных секций `JOIN` в одном запросе `SELECT`:

- Получение всех столбцов через `*` возможно только при объединении таблиц, но не подзапросов.
- Секция `PREWHERE` недоступна.

Для секций `ON`, `WHERE` и `GROUP BY`:

- Нельзя использовать произвольные выражения в секциях `ON`, `WHERE`, и `GROUP BY`, однако можно определить выражение в секции `SELECT` и затем использовать его через алиас в других секциях.

## Производительность

При запуске `JOIN`, отсутствует оптимизация порядка выполнения по отношению к другим стадиям запроса. Соединение (поиск в «правой» таблице) выполняется до фильтрации в `WHERE` и до агрегации. Чтобы явно задать порядок вычислений, рекомендуется выполнять `JOIN` подзапроса с подзапросом.

Каждый раз для выполнения запроса с одинаковым `JOIN`, подзапрос выполняется заново — результат не кэшируется. Это можно избежать, используя специальный движок таблиц `Join`, представляющий собой подготовленное множество для соединения, которое всегда находится в оперативке.

В некоторых случаях это более эффективно использовать `IN` вместо `JOIN`.

Если `JOIN` необходим для соединения с таблицами измерений (dimension tables - сравнительно небольшие таблицы, которые содержат свойства измерений - например, имена для рекламных кампаний), то использование `JOIN` может быть не очень удобным из-за громоздкости синтаксиса, а также из-за того, что правая таблица читается заново при каждом запросе. Специально для таких случаев существует функциональность «Внешние словари», которую следует использовать вместо `JOIN`. Дополнительные сведениясмотрите в разделе «Внешние словари».

## Ограничения по памяти

По умолчанию ClickHouse использует алгоритм `hash join`. ClickHouse берет правую таблицу и создает для нее хеш-таблицу в оперативной памяти. При включённой настройке `join_algorithm = 'auto'`, после некоторого порога потребления памяти ClickHouse переходит к алгоритму `merge join`. Описание алгоритмов `JOIN` см. в настройке `join_algorithm`.

Если вы хотите ограничить потребление памяти во время выполнения операции `JOIN`, используйте настройки:

- `max_rows_in_join` — ограничивает количество строк в хеш-таблице.
- `max_bytes_in_join` — ограничивает размер хеш-таблицы.

По достижении любого из этих ограничений ClickHouse действует в соответствии с настройкой `join_overflow_mode`.

## Примеры

Пример:

```
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

## Секция LIMIT

LIMIT *m* позволяет выбрать из результата первые *m* строк.

LIMIT *n, m* позволяет выбрать из результата первые *m* строк после пропуска первых *n* строк.  
Синтаксис LIMIT *m* OFFSET *n* также поддерживается.

*n* и *m* должны быть неотрицательными целыми числами.

При отсутствии секции ORDER BY, однозначно сортирующей результат, результат может быть произвольным и может являться недетерминированным.

### Примечание

Количество возвращаемых строк может зависеть также от настройки limit.

## Модификатор LIMIT ... WITH TIES

Когда вы установите модификатор WITH TIES для LIMIT *n[,m]* и указываете ORDER BY expr\_list, вы получите первые *n* или *n,m* строк и дополнительно все строки с теми же самыми значениями полей указанных в ORDER BY равными строке на позиции *n* для LIMIT *n* или *m* для LIMIT *n,m*.

Этот модификатор также может быть скомбинирован с ORDER BY ... WITH FILL модификатором

Для примера следующий запрос:

```
SELECT * FROM (
    SELECT number%50 AS n FROM numbers(100)
) ORDER BY n LIMIT 0,5
```

возвращает

n
0
0
1
1
2

но после применения модификатора `WITH TIES`

```
SELECT * FROM (
    SELECT number%50 AS n FROM numbers(100)
) ORDER BY n LIMIT 0,5 WITH TIES
```

возвращает другой набор строк

n
0
0
1
1
2
2

поскольку строка на позиции 6 имеет тоже самое значение "2" для поля `n` что и строка на позиции 5

## Секция `LIMIT BY`

Запрос с секцией `LIMIT n BY expressions` выбирает первые `n` строк для каждого отличного значения `expressions`. Ключ `LIMIT BY` может содержать любое количество **выражений**.

ClickHouse поддерживает следующий синтаксис:

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

Во время обработки запроса, ClickHouse выбирает данные, упорядоченные по ключу сортировки. Ключ сортировки задаётся явно в секции `ORDER BY` или неявно в свойствах движка таблицы. Затем ClickHouse применяет `LIMIT n BY expressions` и возвращает первые `n` для каждой отличной комбинации `expressions`. Если указан `OFFSET`, то для каждого блока данных, который принадлежит отдельной комбинации `expressions`, ClickHouse отступает `offset_value` строк от начала блока и возвращает не более `n`. Если `offset_value` больше, чем количество строк в блоке данных, ClickHouse не возвращает ни одной строки.

`LIMIT BY` не связана с секцией `LIMIT`. Их можно использовать в одном запросе.

Если вы хотите использовать в секции `LIMIT BY` номера столбцов вместо названий, включите настройку `enable_positional_arguments`.

# Примеры

Образец таблицы:

```
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);
```

Запросы:

```
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

Запрос `SELECT * FROM limit_by ORDER BY id, val LIMIT 2 OFFSET 1 BY id` возвращает такой же результат.

Следующий запрос выбирает топ 5 рефереров для каждой пары `domain, device_type`, но не более 100 строк (`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
```

Запрос выберет топ 5 рефереров для каждой пары `domain, device_type`, но не более 100 строк (`LIMIT n BY + LIMIT`).

`LIMIT n BY` работает с `NULL` как если бы это было конкретное значение. Т.е. в результате запроса пользователь получит все комбинации полей, указанных в `BY`.

## Секция OFFSET FETCH

`OFFSET` и `FETCH` позволяют извлекать данные по частям. Они указывают строки, которые вы хотите получить в результате запроса.

```
OFFSET offset_row_count {ROW | ROWS} [FETCH {FIRST | NEXT} fetch_row_count {ROW | ROWS} {ONLY | WITH TIES}]
```

`offset_row_count` или `fetch_row_count` может быть числом или литеральной константой. Если вы не задаете `fetch_row_count` явно, используется значение по умолчанию, равное 1.

`OFFSET` указывает количество строк, которые необходимо пропустить перед началом возврата строк из запроса.

`FETCH` указывает максимальное количество строк, которые могут быть получены в результате запроса.

Опция `ONLY` используется для возврата строк, которые следуют сразу же за строками, пропущенными секцией `OFFSET`. В этом случае `FETCH` — это альтернатива `LIMIT`. Например, следующий запрос

```
SELECT * FROM test_fetch ORDER BY a OFFSET 1 ROW FETCH FIRST 3 ROWS ONLY;
```

идентичен запросу

```
SELECT * FROM test_fetch ORDER BY a LIMIT 3 OFFSET 1;
```

Опция `WITH TIES` используется для возврата дополнительных строк, которые привязываются к последней в результате запроса. Например, если `fetch_row_count` имеет значение 5 и существуют еще 2 строки с такими же значениями столбцов, указанных в `ORDER BY`, что и у пятой строки результата, то финальный набор будет содержать 7 строк.

## Примечание

Секция `OFFSET` должна находиться перед секцией `FETCH`, если обе присутствуют.

## Примечание

Общее количество пропущенных строк может зависеть также от настройки `offset`.

## Примеры

Входная таблица:

a	b
1	1
2	1
3	4
1	3
5	4
0	6
5	7

Использование опции `ONLY`:

```
SELECT * FROM test_fetch ORDER BY a OFFSET 3 ROW FETCH FIRST 3 ROWS ONLY;
```

Результат:

a	b
2	1
3	4
5	4

Использование опции `WITH TIES`:

```
SELECT * FROM test_fetch ORDER BY a OFFSET 3 ROWS FIRST 3 ROWS WITH TIES;
```

Результат:

a	b
2	1
3	4
5	4
5	7

## Секция ORDER BY

Секция `ORDER BY` содержит список выражений, к каждому из которых также может быть приписано `DESC` или `ASC` (направление сортировки). Если ничего не приписано - это аналогично приписыванию `ASC`. `ASC` - сортировка по возрастанию, `DESC` - сортировка по убыванию. Обозначение направления сортировки действует на одно выражение, а не на весь список. Пример: `ORDER BY Visits DESC, SearchPhrase`.

Если вы хотите для сортировки данных указывать номера столбцов, а не названия, включите настройку `enable_positional_arguments`.

Строки, для которых список выражений, по которым производится сортировка, принимает одинаковые значения, выводятся в произвольном порядке, который может быть также недетерминированным (каждый раз разным).

Если секция `ORDER BY` отсутствует, то, аналогично, порядок, в котором идут строки, не определён, и может быть недетерминированным.

## Сортировка специальных значений

Существует два подхода к участию `NaN` и `NULL` в порядке сортировки:

- По умолчанию или с модификатором `NULLS LAST`: сначала остальные значения, затем `NaN`, затем `NULL`.
- С модификатором `NULLS FIRST`: сначала `NULL`, затем `NaN`, затем другие значения.

## Пример

Для таблицы

X	Y
1	NULL
2	2
1	nan
2	2
3	4
5	6
6	nan
7	NULL
6	7
8	9

Выполните запрос `SELECT * FROM t_null_nan ORDER BY y NULLS FIRST` чтобы получить:

X	Y
1	NULL
7	NULL
1	nan
6	nan
2	2
2	2
3	4
5	6
6	7
8	9

При сортировке чисел с плавающей запятой NaNs отделяются от других значений. Независимо от порядка сортировки, NaNs приходят в конце. Другими словами, при восходящей сортировке они помещаются так, как будто они больше всех остальных чисел, а при нисходящей сортировке они помещаются так, как будто они меньше остальных.

## Поддержка collation

Для сортировки по значениям типа `String` есть возможность указать `collation` (сравнение). Пример:  
`ORDER BY SearchPhrase COLLATE 'tr'` - для сортировки по поисковой фразе, по возрастанию, с учётом турецкого алфавита, регистронезависимо, при допущении, что строки в кодировке UTF-8. `COLLATE` может быть указан или не указан для каждого выражения в `ORDER BY` независимо. Если есть `ASC` или `DESC`, то `COLLATE` указывается после них. При использовании `COLLATE` сортировка всегда регистронезависима.

Сравнение поддерживается при использовании типов `LowCardinality`, `Nullable`, `Array` и `Tuple`.

Рекомендуется использовать `COLLATE` только для окончательной сортировки небольшого количества строк, так как производительность сортировки с указанием `COLLATE` меньше, чем обычной сортировки по байтам.

## Примеры с использованием сравнения

Пример с значениями типа `String`:

Входная таблица:

X	S
1	bca
2	ABC
3	123a
4	abc
5	BCA

Запрос:

```
SELECT * FROM collate_test ORDER BY s ASC COLLATE 'en';
```

Результат:

X	S
3	123a
4	abc
2	ABC
1	bca
5	BCA

Пример со строками типа **Nullable**:

Входная таблица:

X	S
1	bca
2	NULL
3	ABC
4	123a
5	abc
6	NULL
7	BCA

Запрос:

```
SELECT * FROM collate_test ORDER BY s ASC COLLATE 'en';
```

Результат:

X	S
4	123a
5	abc
3	ABC
1	bca
7	BCA
6	NULL
2	NULL

Пример со строками в **Array**:

Входная таблица:

X	S
1	['Z']
2	['z']
3	['a']
4	['A']
5	['z','a']
6	['z','a','a']
7	[]

Запрос:

```
SELECT * FROM collate_test ORDER BY s ASC COLLATE 'en';
```

Результат:

X	S
7	['']
3	['a']
4	['A']
2	['z']
5	['z','a']
6	['z','a','a']
1	['Z']

Пример со строками типа **LowCardinality**:

Входная таблица:

X	S
1	Z
2	z
3	a
4	A
5	za
6	zaa
7	

Запрос:

```
SELECT * FROM collate_test ORDER BY s ASC COLLATE 'en';
```

Результат:

X	S
7	
3	a
4	A
2	z
1	Z
5	za
6	zaa

Пример со строками в **Tuple**:

X	S
1	(1,'Z')
2	(1,'z')
3	(1,'a')
4	(2,'z')
5	(1,'A')
6	(2,'Z')
7	(2,'A')

Запрос:

```
SELECT * FROM collate_test ORDER BY s ASC COLLATE 'en';
```

Результат:

X	S
3	(1,'a')
5	(1,'A')
2	(1,'z')
1	(1,'Z')
7	(2,'A')
4	(2,'z')
6	(2,'Z')

## Деталь реализации

Если кроме `ORDER BY` указан также не слишком большой `LIMIT`, то расходуется меньше оперативки. Иначе расходуется количество памяти, пропорциональное количеству данных для сортировки. При распределённой обработке запроса, если отсутствует `GROUP BY`, сортировка частично делается на удалённых серверах, а на сервере-инициаторе запроса производится слияние результатов. Таким образом, при распределённой сортировке, может сортироваться объём данных, превышающий размер памяти на одном сервере.

Существует возможность выполнять сортировку во внешней памяти (с созданием временных файлов на диске), если оперативной памяти не хватает. Для этого предназначена настройка `max_bytes_before_external_sort`. Если она выставлена в 0 (по умолчанию), то внешняя сортировка выключена. Если она включена, то при достижении объёма данных для сортировки указанного количества байт, накопленные данные будут отсортированы и сброшены во временный файл. После того, как все данные будут прочитаны, будет произведено слияние всех сортированных файлов и выдача результата. Файлы записываются в директорию `/var/lib/clickhouse/tmp/` (по умолчанию, может быть изменено с помощью параметра `tmp_path`) в конфиге.

На выполнение запроса может расходоваться больше памяти, чем `max_bytes_before_external_sort`. Поэтому, значение этой настройки должно быть существенно меньше, чем `max_memory_usage`. Для примера, если на вашем сервере 128 GB оперативки, и вам нужно выполнить один запрос, то выставите `max_memory_usage` в 100 GB, а `max_bytes_before_external_sort` в 80 GB.

Внешняя сортировка работает существенно менее эффективно, чем сортировка в оперативке.

## Оптимизация чтения данных

Если в списке выражений в секции `ORDER BY` первыми указаны те поля, по которым проиндексирована таблица, по которой строится выборка, такой запрос можно оптимизировать — для этого используйте настройку `optimize_read_in_order`.

Когда настройка `optimize_read_in_order` включена, при выполнении запроса сервер использует табличные индексы и считывает данные в том порядке, который задан списком выражений `ORDER BY`. Поэтому если в запросе установлен `LIMIT`, сервер не станет считывать лишние данные. Таким образом, запросы к большим таблицам, но имеющие ограничения по числу записей, выполняются быстрее.

Оптимизация работает при любом порядке сортировки `ASC` или `DESC`, но не работает при использовании группировки `GROUP BY` и модификатора `FINAL`.

Когда настройка `optimize_read_in_order` отключена, при выполнении запросов `SELECT` табличные индексы не используются.

Для запросов с сортировкой `ORDER BY`, большим значением `LIMIT` и условиями отбора `WHERE`, требующими чтения больших объемов данных, рекомендуется отключать `optimize_read_in_order` вручную.

Оптимизация чтения данных поддерживается в следующих движках:

- [MergeTree](#)
- [Merge](#), [Buffer](#) и [MaterializedView](#), работающими с таблицами MergeTree

В движке [MaterializedView](#) оптимизация поддерживается при работе с сохраненными запросами (представлениями) вида `SELECT ... FROM merge_tree_table ORDER BY pk`. Но оптимизация не поддерживается для запросов вида `SELECT ... FROM view ORDER BY pk`, если в сохраненном запросе нет секции `ORDER BY`.

## Модификатор ORDER BY expr WITH FILL

Этот модификатор также может быть скобинирован с модификатором [LIMIT ... WITH TIES](#)

Модификатор `WITH FILL` может быть установлен после `ORDER BY expr` с опциональными параметрами `FROM expr`, `TO expr` и `STEP expr`.

Все пропущенные значения для колонки `expr` будут заполнены значениями, соответствующими предполагаемой последовательности значений колонки, другие колонки будут заполнены значениями по умолчанию.

Используйте следующую конструкцию для заполнения нескольких колонок с модификатором `WITH FILL` с необязательными параметрами после каждого имени поля в секции `ORDER BY`.

```
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` может быть применен к полям с числовыми (все разновидности `float`, `int`, `decimal`) или временными (все разновидности `Date`, `DateTime`) типами. В случае применения к полям типа `String` недостающие значения заполняются пустой строкой.

Когда не определен `FROM const_expr`, последовательность заполнения использует минимальное значение поля `expr` из `ORDER BY`.

Когда не определен `TO const_expr`, последовательность заполнения использует максимальное значение поля `expr` из `ORDER BY`.

Когда `STEP const_numeric_expr` определен, `const_numeric_expr` интерпретируется "как есть" для числовых типов, как "дни" для типа `Date` и как "секунды" для типа `DateTime`.

Когда `STEP const_numeric_expr` не указан, тогда используется 1.0 для числовых типов, 1 день для типа `Date` и 1 секунда для типа `DateTime`.

Пример запроса без использования `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;
```

Результат:

n	source
1	original
4	original
7	original

Тот же запрос после применения модификатора `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 WITH FILL FROM 0 TO 5.51 STEP 0.5

```

Результат:

n	source
0	
0.5	
1	original
1.5	
2	
2.5	
3	
3.5	
4	original
4.5	
5	
5.5	
7	original

Для случая с несколькими полями ORDER BY field2 WITH FILL, field1 WITH FILL порядок заполнения будет соответствовать порядку полей в секции ORDER BY.

Пример:

```

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;

```

Результат:

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

Поле d1 не заполняется и использует значение по умолчанию. Поскольку у нас нет повторяющихся значений для d2, мы не можем правильно рассчитать последовательность заполнения для d1.

Следующий запрос (с измененным порядком в 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;

```

Результат:

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

## Секция PREWHERE

Prewhere — это оптимизация для более эффективного применения фильтрации. Она включена по умолчанию, даже если секция PREWHERE явно не указана. В этом случае работает автоматическое перемещение части выражения из WHERE до стадии prewhere. Роль секции PREWHERE только для управления этой оптимизацией, если вы думаете, что знаете, как сделать перемещение условия лучше, чем это происходит по умолчанию.

При оптимизации prewhere сначала читаются только те столбцы, которые необходимы для выполнения выражения prewhere. Затем читаются другие столбцы, необходимые для выполнения остальной части запроса, но только те блоки, в которых находится выражение prewhere «верно» по крайней мере для некоторых рядов. Если есть много блоков, где выражение prewhere «ложно» для всех строк и для выражения prewhere требуется меньше столбцов, чем для других частей запроса, это часто позволяет считывать гораздо меньше данных с диска для выполнения запроса.

## Управление PREWHERE вручную

PREWHERE имеет смысл использовать, если есть условия фильтрации, которые использует меньшинство столбцов из тех, что есть в запросе, но достаточно сильно фильтрует данные. Таким образом, сокращается количество читаемых данных.

В запросе может быть одновременно указаны и PREWHERE, и WHERE. В этом случае PREWHERE предшествует WHERE.

Если значение параметра `optimize_move_to_prewhere` равно 0, эвристика по автоматическому перемещению части выражений из WHERE к PREWHERE отключается.

Если в запросе есть модификатор FINAL, оптимизация PREWHERE не всегда корректна. Она действует только если включены обе настройки `optimize_move_to_prewhere` и `optimize_move_to_prewhere_if_final`.

## Внимание

Секция PREWHERE выполняется до FINAL, поэтому результаты запросов FROM ... FINAL могут исказиться при использовании PREWHERE с полями, не входящими в ORDER BY таблицы.

## Ограничения

PREWHERE поддерживается только табличными движками из семейства \*MergeTree.

# Секция SAMPLE

Секция SAMPLE позволяет выполнять запросы приближённо. Например, чтобы посчитать статистику по всем визитам, можно обработать 1/10 всех визитов и результат умножить на 10.

Сэмплирование имеет смысл, когда:

1. Точность результата не важна, например, для оценочных расчетов.
2. Возможности аппаратной части не позволяют соответствовать строгим критериям. Например, время ответа должно быть <100 мс. При этом точность расчета имеет более низкий приоритет.
3. Точность результата участует в бизнес-модели сервиса. Например, пользователи с бесплатной подпиской на сервис могут получать отчеты с меньшей точностью, чем пользователи с премиум подпиской.

## Внимание

Не стоит использовать сэмплирование в тех задачах, где важна точность расчетов. Например, при работе с финансовыми отчетами.

Свойства сэмплирования:

- Сэмплирование работает детерминированно. При многократном выполнении одного и того же запроса `SELECT .. SAMPLE`, результат всегда будет одинаковым.
- Сэмплирование поддерживает консистентность для разных таблиц. Имеется в виду, что для таблиц с одним и тем же ключом сэмплирования, подмножество данных в выборках будет одинаковым (выборки при этом должны быть сформированы для одинаковой доли данных). Например, выборка по идентификаторам посетителей выберет из разных таблиц строки с одинаковым подмножеством всех возможных идентификаторов. Это свойство позволяет использовать выборки в подзапросах в секции `IN`, а также объединять выборки с помощью `JOIN`.
- Сэмплирование позволяет читать меньше данных с диска. Обратите внимание, для этого необходимо корректно указать ключ сэмплирования. Подробнее см. в разделе [Создание таблицы MergeTree](#).

Сэмплирование поддерживается только таблицами семейства `MergeTree` и только в том случае, если для таблиц был указан ключ сэмплирования (выражение, на основе которого должна производиться выборка). Подробнее см. в разделе [Создание таблиц MergeTree](#).

Выражение SAMPLE в запросе можно задать следующими способами:

Способ задания SAMPLE	Описание
<code>SAMPLE k</code>	Здесь <code>k</code> – это дробное число в интервале от 0 до 1. Запрос будет выполнен по <code>k</code> доле данных. Например, если указано <code>SAMPLE 1/10</code> , то запрос будет выполнен для выборки из 1/10 данных. <a href="#">Подробнее</a>
<code>SAMPLE n</code>	Здесь <code>n</code> – это достаточно большое целое число.

Запрос будет выполнен для выборки, состоящей из не менее чем `n` строк. Например, если указано `SAMPLE 10000000`, то запрос будет выполнен для не менее чем 10,000,000 строк. [Подробнее](#) `SAMPLE k` `OFFSET m` Здесь `k` и `m` – числа от 0 до 1. Запрос будет выполнен по `k` доле данных. При этом выборка

будет сформирована со смещением на  $m$  долю. Подробнее

## SAMPLE k

Здесь  $k$  – число в интервале от 0 до 1. Поддерживается как дробная, так и десятичная форма записи. Например, SAMPLE 1/2 или SAMPLE 0.5.

Если задано выражение SAMPLE k, запрос будет выполнен для  $k$  доли данных. Рассмотрим пример:

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

В этом примере запрос выполняется по выборке из 0.1 (10%) данных. Значения агрегатных функций не корректируются автоматически, поэтому чтобы получить приближённый результат, значение count() нужно вручную умножить на 10.

Выборка с указанием относительного коэффициента является «согласованной»: для таблиц с одним и тем же ключом сэмплирования, выборка с одинаковой относительной долей всегда будет составлять одно и то же подмножество данных. То есть выборка из разных таблиц, на разных серверах, в разное время, формируется одинаковым образом.

## SAMPLE n

Здесь  $n$  – это достаточно большое целое число. Например, SAMPLE 10000000.

Если задано выражение SAMPLE n, запрос будет выполнен для выборки из не менее  $n$  строк (но не значительно больше этого значения). Например, если задать SAMPLE 10000000, в выборку попадут не менее 10,000,000 строк.

### Примечание

Следует иметь в виду, что  $n$  должно быть достаточно большим числом. Так как минимальной единицей данных для чтения является одна гранула (её размер задаётся настройкой index\_granularity для таблицы), имеет смысл создавать выборки, размер которых существенно превосходит размер гранулы.

При выполнении SAMPLE n коэффициент сэмплирования заранее неизвестен (то есть нет информации о том, относительно какого количества данных будет сформирована выборка). Чтобы узнать коэффициент сэмплирования, используйте столбец `_sample_factor`.

Виртуальный столбец `_sample_factor` автоматически создается в тех таблицах, для которых задано выражение SAMPLE BY (подробнее см. в разделе [Создание таблицы MergeTree](#)). В столбце содержится коэффициент сэмплирования для таблицы – он рассчитывается динамически по мере добавления данных в таблицу. Ниже приведены примеры использования столбца `_sample_factor`.

Предположим, у нас есть таблица, в которой ведется статистика посещений сайта. Пример ниже показывает, как рассчитать суммарное число просмотров:

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

Следующий пример показывает, как посчитать общее число визитов:

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

В примере ниже рассчитывается среднее время на сайте. Обратите внимание, при расчете средних значений, умножать результат на коэффициент сэмплирования не нужно.

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

## SAMPLE k OFFSET m

Здесь  $k$  и  $m$  – числа в интервале от 0 до 1. Например, SAMPLE 0.1 OFFSET 0.5. Поддерживается как дробная, так и десятичная форма записи.

При задании SAMPLE k OFFSET m, выборка будет сформирована из  $k$  доли данных со смещением на долю  $m$ . Примеры приведены ниже.

### Пример 1

```
SAMPLE 1/10
```

В этом примере выборка будет сформирована по 1/10 доле всех данных:

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

### Пример 2

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

Здесь выборка, которая состоит из 1/10 доли данных, взята из второй половины данных.

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

## Секция UNION

Вы можете использовать UNION в двух режимах: UNION ALL или UNION DISTINCT.

Если UNION используется без указания ALL или DISTINCT, то его поведение определяется настройкой `union_default_mode`. Разница между UNION ALL и UNION DISTINCT в том, что UNION DISTINCT выполняет явное преобразование для результата объединения. Это равнозначно выражению `SELECT DISTINCT` из подзапроса, содержащего UNION ALL.

Чтобы объединить любое количество SELECT запросов путем объединения их результатов, вы можете использовать UNION. Пример:

```
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
```

Результирующие столбцы сопоставляются по их индексу (порядку внутри SELECT). Если имена столбцов не совпадают, то имена для конечного результата берутся из первого запроса.

При объединении выполняет приведение типов. Например, если два запроса имеют одно и то же поле с не-Nullable и Nullable совместимыми типами, полученные в результате UNION данные будут иметь Nullable тип.

Запросы, которые являются частью UNION, могут быть заключены в круглые скобки. **ORDER BY** и **LIMIT** применяются к отдельным запросам, а не к конечному результату. Если вам нужно применить преобразование к конечному результату, вы можете разместить все объединенные с помощью UNION запросы в подзапрос в секции **FROM**.

Если используете UNION без явного указания UNION ALL или UNION DISTINCT, то вы можете указать режим объединения с помощью настройки `union_default_mode`, значениями которой могут быть ALL, DISTINCT или пустая строка. Однако если вы используете UNION с настройкой `union_default_mode`, значением которой является пустая строка, то будет сгенерировано исключение. В следующих примерах продемонстрированы результаты запросов при разных значениях настройки.

Запрос:

```
SET union_default_mode = 'DISTINCT';
SELECT 1 UNION SELECT 2 UNION SELECT 3 UNION SELECT 2;
```

Результат:

1
1
1
2
2
1
3
3

Запрос:

```
SET union_default_mode = 'ALL';
SELECT 1 UNION SELECT 2 UNION SELECT 3 UNION SELECT 2;
```

Результат:

1
1

1
2

1
2

1
3

Запросы, которые являются частью UNION/UNION ALL/UNION DISTINCT, выполняются параллельно, и их результаты могут быть смешаны вместе.

### Смотрите также

- Настройка [insert\\_null\\_as\\_default](#).
- Настройка [union\\_default\\_mode](#).

## Секция WHERE

Позволяет задать выражение, которое ClickHouse использует для фильтрации данных перед всеми другими действиями в запросе кроме выражений, содержащихся в секции [PREWHERE](#). Обычно, это выражение с логическими операторами.

Результат выражения должен иметь тип UInt8.

ClickHouse использует в выражении индексы, если это позволяет [движок таблицы](#).

Если в секции необходимо проверить [NULL](#), то используйте операторы [IS NULL](#) и [IS NOT NULL](#), а также соответствующие функции [isNull](#) и [isNotNull](#). В противном случае выражение будет считаться всегда не выполненным.

Пример проверки на [NULL](#):

```
SELECT * FROM t_null WHERE y IS NULL
```

x		y
1	NULL	

### Примечание

Существует оптимизация фильтрации под названием [prewhere](#).

## Секция WITH

Clickhouse поддерживает [Общие табличные выражения](#), то есть позволяет использовать результаты выражений из секции [WITH](#) в остальной части [SELECT](#) запроса. Именованные подзапросы могут быть включены в текущий и дочерний контекст запроса в тех местах, где разрешены табличные объекты. Рекурсия предотвращается путем скрытия общего табличного выражения текущего уровня из выражения [WITH](#).

# Синтаксис

```
WITH <expression> AS <identifier>
```

или

```
WITH <identifier> AS <subquery expression>
```

## Примеры

### Пример 1: Использование константного выражения как «переменной»

```
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;
```

### Пример 2: Выкидывание выражения sum(bytes) из списка колонок в SELECT

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

### Пример 3: Использование результатов скалярного подзапроса

```
/* запрос покажет TOP 10 самых больших таблиц */
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;
```

### Пример 4: Переиспользование выражения

```
WITH test1 AS (SELECT i + 1, j + 1 FROM test1)
SELECT * FROM test1;
```

# INSERT

Добавление данных.

Базовый формат запроса:

```
INSERT INTO [db.]table [(c1, c2, c3)] VALUES (v11, v12, v13), (v21, v22, v23), ...
```

Вы можете указать список столбцов для вставки, используя синтаксис (c1, c2, c3). Также можно использовать выражение со **звездочкой** и/или модификаторами, такими как **APPLY**, **EXCEPT**, **REPLACE**.

В качестве примера рассмотрим таблицу:

```
SHOW CREATE insert_select_testtable
```

```
statement
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)
```

Если вы хотите вставить данные во все столбцы, кроме 'b', вам нужно передать столько значений, сколько столбцов вы указали в скобках:

```
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

В этом примере мы видим, что вторая строка содержит столбцы a и c, заполненные переданными значениями и b, заполненный значением по умолчанию.

Если список столбцов не включает все существующие столбцы, то все остальные столбцы заполняются следующим образом:

- Значения, вычисляемые из **DEFAULT** выражений, указанных в определении таблицы.
- Нули и пустые строки, если **DEFAULT** не определены.

В **INSERT** можно передавать данные любого **формата**, который поддерживает ClickHouse. Для этого формат необходимо указать в запросе в явном виде:

```
INSERT INTO [db.]table [(c1, c2, c3)] FORMAT format_name data_set
```

Например, следующий формат запроса идентичен базовому варианту **INSERT ... VALUES**:

```
INSERT INTO [db.]table [(c1, c2, c3)] FORMAT Values (v11, v12, v13), (v21, v22, v23), ...
```

ClickHouse отсекает все пробелы и один перенос строки (если он есть) перед данными. Рекомендуем при формировании запроса переносить данные на новую строку после операторов запроса (это важно, если данные начинаются с пробелов).

Пример:

```
INSERT INTO t FORMAT TabSeparated
11 Hello, world!
22 Qwerty
```

С помощью консольного клиента или HTTP интерфейса можно вставлять данные отдельно от запроса. Как это сделать, читайте в разделе «[Интерфейсы](#)».

## Ограничения (constraints)

Если в таблице объявлены [ограничения](#), то их выполнимость будет проверена для каждой вставляемой строки. Если для хотя бы одной строки ограничения не будут выполнены, запрос будет остановлен.

## Вставка результатов SELECT

```
INSERT INTO [db.]table [(c1, c2, c3)] SELECT ...
```

Соответствие столбцов определяется их позицией в секции `SELECT`. При этом, их имена в выражении `SELECT` и в таблице для `INSERT`, могут отличаться. При необходимости выполняется приведение типов данных, эквивалентное соответствующему оператору `CAST`.

Все форматы данных кроме `Values` не позволяют использовать в качестве значений выражения, такие как `now()`, `1 + 2` и подобные. Формат `Values` позволяет ограниченно использовать выражения, но это не рекомендуется, так как в этом случае для их выполнения используется неэффективный вариант кода.

Не поддерживаются другие запросы на модификацию части данных: `UPDATE`, `DELETE`, `REPLACE`, `MERGE`, `UPSERT`, `INSERT UPDATE`.

Вы можете удалять старые данные с помощью запроса `ALTER TABLE ... DROP PARTITION`.

Для табличной функции `input()` после секции `SELECT` должна следовать секция `FORMAT`.

Чтобы вставить значение по умолчанию вместо `NULL` в столбец, который не позволяет хранить `NULL`, включите настройку `insert_null_as_default`.

## Замечания о производительности

`INSERT` сортирует входящие данные по первичному ключу и разбивает их на партиции по ключу парitionирования. Если вы вставляете данные в несколько партиций одновременно, то это может значительно снизить производительность запроса `INSERT`. Чтобы избежать этого:

- Добавляйте данные достаточно большими пачками. Например, по 100 000 строк.
- Группируйте данные по ключу парitionирования самостоятельно перед загрузкой в ClickHouse.

Снижения производительности не будет, если:

- Данные поступают в режиме реального времени.

- Вы загружаете данные, которые как правило отсортированы по времени.

Также возможно вставлять данные асинхронно во множественных маленьких вставках. Данные от таких вставок сначала собираются в пачки, а потом вставляются в таблицу. Чтобы включить асинхронный режим, используйте настройку [async\\_insert](#). Обратите внимание, что асинхронные вставки поддерживаются только через протокол HTTP, а дедупликация при этом не производится.

## См. также

- [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](#)
- 

## Запросы CREATE

Запрос create создает новую сущность одного из следующих типов:

- [DATABASE](#)
  - [TABLE](#)
  - [VIEW](#)
  - [DICTIONARY](#)
  - [FUNCTION](#)
  - [USER](#)
  - [ROLE](#)
  - [ROW POLICY](#)
  - [QUOTA](#)
  - [SETTINGS PROFILE](#)
- 

## CREATE DATABASE

Создает базу данных.

```
CREATE DATABASE [IF NOT EXISTS] db_name [ON CLUSTER cluster] [ENGINE = engine(...)]
```

## Секции

#### ■ IF NOT EXISTS

Если база данных с именем `db\_name` уже существует, то ClickHouse не создаёт базу данных и:

- Не генерирует исключение, если секция указана.
- Генерирует исключение, если секция не указана.

#### ■ ON CLUSTER

ClickHouse создаёт базу данных `db\_name` на всех серверах указанного кластера.

#### ■ ENGINE

- MySQL

Позволяет получать данные с удаленного сервера MySQL.

По умолчанию ClickHouse использует собственный движок баз данных.

## CREATE TABLE

Запрос `CREATE TABLE` может иметь несколько форм, которые используются в зависимости от контекста и решаемых задач.

По умолчанию таблицы создаются на текущем сервере. Распределенные DDL запросы создаются с помощью секции `ON CLUSTER`, которая [описана отдельно](#).

### Варианты синтаксиса

#### С описанием структуры

```
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
```

Создаёт таблицу с именем name в БД db или текущей БД, если db не указана, со структурой, указанной в скобках, и движком engine.

Структура таблицы представляет список описаний столбцов. Индексы, если поддерживаются движком, указываются в качестве параметров для движка таблицы.

Описание столбца, это `name type`, в простейшем случае. Пример: `RegionID UInt32`.

Также могут быть указаны выражения для значений по умолчанию - смотрите ниже.

При необходимости можно указать [первичный ключ](#) с одним или несколькими ключевыми выражениями.

#### Со структурой, аналогичной другой таблице

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name AS [db2.]name2 [ENGINE = engine]
```

Создаёт таблицу с такой же структурой, как другая таблица. Можно указать другой движок для таблицы. Если движок не указан, то будет выбран такой же движок, как у таблицы `db2.name2`.

#### Из табличной функции

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name AS table_function()
```

Создаёт таблицу с такой же структурой и данными, как результат соответствующей табличной функции. Созданная таблица будет работать так же, как и указанная табличная функция.

## Из запроса SELECT

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name[(name1 [type1], name2 [type2], ...)] ENGINE = engine AS SELECT ...
```

Создаёт таблицу со структурой, как результат запроса `SELECT`, с движком `engine`, и заполняет её данными из `SELECT`. Также вы можете явно задать описание столбцов.

Если таблица уже существует и указано `IF NOT EXISTS`, то запрос ничего не делает.

После секции `ENGINE` в запросе могут использоваться и другие секции в зависимости от движка. Подробную документацию по созданию таблиц смотрите в описаниях [движков таблиц](#).

### Пример

Запрос:

```
CREATE TABLE t1 (x String) ENGINE = Memory AS SELECT 1;
SELECT x, toTypeName(x) FROM t1;
```

Результат:

x	toTypeName(x)
1	String

## Модификатор NULL или NOT NULL

Модификатор `NULL` или `NOT NULL`, указанный после типа данных в определении столбца, позволяет или не позволяет типу данных быть [Nullable](#).

Если тип не `Nullable` и указан модификатор `NULL`, то столбец будет иметь тип `Nullable`; если `NOT NULL`, то не `Nullable`. Например, `INT NULL` то же, что и `Nullable(INT)`. Если тип `Nullable` и указаны модификаторы `NULL` или `NOT NULL`, то будет вызвано исключение.

Смотрите также настройку [data\\_type\\_default\\_nullable](#).

## Значения по умолчанию

В описании столбца, может быть указано выражение для значения по умолчанию, одного из следующих видов:

`DEFAULT expr`, `MATERIALIZED expr`, `ALIAS expr`.

Пример: `URLDomain String DEFAULT domain(URL)`.

Если выражение для значения по умолчанию не указано, то в качестве значений по умолчанию будут использоваться нули для чисел, пустые строки для строк, пустые массивы для массивов, а также `0000-00-00` для дат и `0000-00-00 00:00:00` для дат с временем. `NONE`-ы не поддерживаются.

В случае, если указано выражение по умолчанию, то указание типа столбца не обязательно. При отсутствии явно указанного типа, будет использован тип выражения по умолчанию. Пример: `EventDate DEFAULT toDate(EventTime)` - для столбца `EventDate` будет использован тип `Date`.

При наличии явно указанного типа данных и выражения по умолчанию, это выражение будет приводиться к указанному типу с использованием функций приведения типа. Пример: `Hits UInt32 DEFAULT 0` - имеет такой же смысл, как `Hits UInt32 DEFAULT toUInt32(0)`.

В качестве выражения для умолчания, может быть указано произвольное выражение от констант и столбцов таблицы. При создании и изменении структуры таблицы, проверяется, что выражения не содержат циклов. При INSERT-е проверяется разрешимость выражений - что все столбцы, из которых их можно вычислить, переданы.

## DEFAULT

`DEFAULT expr`

Обычное значение по умолчанию. Если в запросе INSERT не указан соответствующий столбец, то он будет заполнен путём вычисления соответствующего выражения.

## MATERIALIZED

`MATERIALIZED expr`

Материализованное выражение. Такой столбец не может быть указан при INSERT, то есть, он всегда вычисляется.

При INSERT без указания списка столбцов, такие столбцы не рассматриваются.

Также этот столбец не подставляется при использовании звёздочки в запросе SELECT. Это необходимо, чтобы сохранить инвариант, что дамп, полученный путём `SELECT *`, можно вставить обратно в таблицу INSERT-ом без указания списка столбцов.

## ALIAS

`ALIAS expr`

Синоним. Такой столбец вообще не хранится в таблице.

Его значения не могут быть вставлены в таблицу, он не подставляется при использовании звёздочки в запросе SELECT.

Он может быть использован в SELECT-ах - в таком случае, во время разбора запроса, алиас раскрывается.

При добавлении новых столбцов с помощью запроса ALTER, старые данные для этих столбцов не записываются. Вместо этого, при чтении старых данных, для которых отсутствуют значения новых столбцов, выполняется вычисление выражений по умолчанию на лету. При этом, если выполнение выражения требует использования других столбцов, не указанных в запросе, то эти столбцы будут дополнительно прочитаны, но только для тех блоков данных, для которых это необходимо.

Если добавить в таблицу новый столбец, а через некоторое время изменить его выражение по умолчанию, то используемые значения для старых данных (для данных, где значения не хранились на диске) поменяются. Также заметим, что при выполнении фоновых слияний, данные для столбцов, отсутствующих в одном из сливающихся кусков, записываются в объединённый кусок.

Отсутствует возможность задать значения по умолчанию для элементов вложенных структур данных.

## Первичный ключ

Вы можете определить [первичный ключ](#) при создании таблицы. Первичный ключ может быть указан двумя способами:

- в списке столбцов:

```
CREATE TABLE db.table_name
(
    name1 type1, name2 type2, ...,
    PRIMARY KEY(expr1[, expr2,...])]
)
ENGINE = engine;
```

- вне списка столбцов:

```
CREATE TABLE db.table_name
(
    name1 type1, name2 type2, ...
)
ENGINE = engine
PRIMARY KEY(expr1[, expr2,...]);
```

## Предупреждение

Вы не можете сочетать оба способа в одном запросе.

## Ограничения

Наряду с объявлением столбцов можно объявить ограничения на значения в столбцах таблицы:

```
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` может быть любым булевым выражением, состоящим из операторов сравнения или функций. При наличии одного или нескольких ограничений в момент вставки данных выражения ограничений будут проверяться на истинность для каждой вставляемой строки данных. В случае, если в теле `INSERT` запроса придут некорректные данные — клиент получит исключение с описанием нарушенного ограничения.

Добавление большого числа ограничений может негативно повлиять на производительность `INSERT` запросов.

## Выражение для TTL

Определяет время хранения значений. Может быть указано только для таблиц семейства `MergeTree`. Подробнее смотрите в [TTL для столбцов и таблиц](#).

## Кодеки сжатия столбцов

По умолчанию, ClickHouse применяет к столбцу метод сжатия, определённый в [конфигурации сервера](#). Кроме этого, можно задать метод сжатия для каждого отдельного столбца в запросе `CREATE TABLE`.

```
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>
...
```

Если кодек `Default` задан для столбца, используется сжатие по умолчанию, которое может зависеть от различных настроек (и свойств данных) во время выполнения.

Пример: `value UInt64 CODEC(Default)` — то же самое, что не указать кодек.

Также можно подменить кодек столбца сжатием по умолчанию, определенным в `config.xml`:

```
ALTER TABLE codec_example MODIFY COLUMN float_value CODEC(Default);
```

Кодеки можно последовательно комбинировать, например, `CODEC(Delta, Default)`.

## Предупреждение

Нельзя распаковать базу данных ClickHouse с помощью сторонних утилит наподобие `lz4`. Необходимо использовать специальную утилиту [clickhouse-compressor](#).

Сжатие поддерживается для следующих движков таблиц:

- [MergeTree family](#)
- [Log family](#)
- [Set](#)
- [Join](#)

ClickHouse поддерживает кодеки общего назначения и специализированные кодеки.

## Кодеки общего назначения

Кодеки:

- `NONE` — без сжатия.
- `LZ4` — [алгоритм сжатия без потерь](#) используемый по умолчанию. Применяет быстрое сжатие LZ4.
- `LZ4HC[(level)]` — алгоритм LZ4 HC (high compression) с настраиваемым уровнем сжатия. Уровень по умолчанию — 9. Настройка `level <= 0` устанавливает уровень сжатия по умолчанию. Возможные уровни сжатия: [1, 12]. Рекомендуемый диапазон уровней: [4, 9].
- `ZSTD[(level)]` — [алгоритм сжатия ZSTD](#) с настраиваемым уровнем сжатия `level`. Возможные уровни сжатия: [1, 22]. Уровень сжатия по умолчанию: 1.

Высокие уровни сжатия полезны для ассимметричных сценариев, подобных «один раз сжал, много раз распаковал». Они подразумевают лучшее сжатие, но большее использование CPU.

## Специализированные кодеки

Эти кодеки разработаны для того, чтобы, используя особенности данных сделать сжатие более эффективным. Некоторые из этих кодеков не сжимают данные самостоятельно. Они готовят данные для кодеков общего назначения, которые сжимают подготовленные данные эффективнее, чем неподготовленные.

Специализированные кодеки:

- `Delta(delta_bytes)` — Метод, в котором исходные значения заменяются разностью двух соседних значений, за исключением первого значения, которое остаётся неизменным. Для хранения разниц используется до `delta_bytes`, т.е. `delta_bytes` — это максимальный размер исходных данных. Возможные значения `delta_bytes`: 1, 2, 4, 8. Значение по умолчанию для `delta_bytes` равно `sizeof(type)`, если результат 1, 2, 4, or 8. Во всех других случаях — 1.
- `DoubleDelta` — Вычисляется разницу от разниц и сохраняет её в компактном бинарном виде. Оптимальная степень сжатия достигается для монотонных последовательностей с постоянным шагом, наподобие временных рядов. Можно использовать с любым типом данных фиксированного размера. Реализует алгоритм, используемый в TSDB Gorilla, поддерживает 64-битные типы данных. Использует 1 дополнительный бит для 32-байтовых значений: 5-битные префиксы вместо 4-битных префиксов. Подробнее читайте в разделе «Compressing Time Stamps» документа [Gorilla: A Fast, Scalable, In-Memory Time Series Database](#).
- `Gorilla` — Вычисляет XOR между текущим и предыдущим значением и записывает результат в компактной бинарной форме. Эффективно сохраняет ряды медленно изменяющихся чисел с плавающей запятой, поскольку наилучший коэффициент сжатия достигается, если соседние значения одинаковые. Реализует алгоритм, используемый в TSDB Gorilla, адаптируя его для работы с 64-битными значениями. Подробнее читайте в разделе «Compressing Values» документа [Gorilla: A Fast, Scalable, In-Memory Time Series Database](#).
- `T64` — Метод сжатия который обрезает неиспользуемые старшие биты целочисленных значений (включая `Enum`, `Date` и `DateTime`). На каждом шаге алгоритма, кодек помещает блок из 64 значений в матрицу  $64 \times 64$ , транспонирует её, обрезает неиспользуемые биты, а то, что осталось возвращает в виде последовательности. Неиспользуемые биты, это биты, которые не изменяются от минимального к максимальному на всём диапазоне значений куска данных.

Кодеки `DoubleDelta` и `Gorilla` используются в TSDB Gorilla как компоненты алгоритма сжатия. Подход `Gorilla` эффективен в сценариях, когда данные представляют собой медленно изменяющиеся во времени величины. Метки времени эффективно сжимаются кодеком `DoubleDelta`, а значения кодеком `Gorilla`. Например, чтобы создать эффективно хранящуюся таблицу, используйте следующую конфигурацию:

```
CREATE TABLE codec_example
(
    timestamp DateTime CODEC(DoubleDelta),
    slow_values Float32 CODEC(Gorilla)
)
ENGINE = MergeTree()
```

## Временные таблицы

ClickHouse поддерживает временные таблицы со следующими характеристиками:

- Временные таблицы исчезают после завершения сессии, в том числе при обрыве соединения.
- Временная таблица использует только модуль памяти.
- Невозможно указать базу данных для временной таблицы. Она создается вне баз данных.

- Невозможно создать временную таблицу распределенным DDL запросом на всех серверах кластера (с опцией `ON CLUSTER`): такая таблица существует только в рамках существующей сессии.
- Если временная таблица имеет то же имя, что и некоторая другая, то, при упоминании в запросе без указания БД, будет использована временная таблица.
- При распределённой обработке запроса, используемые в запросе временные таблицы, передаются на удалённые серверы.

Чтобы создать временную таблицу, используйте следующий синтаксис:

```
CREATE TEMPORARY TABLE [IF NOT EXISTS] table_name
(
    name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1],
    name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2],
    ...
)
```

В большинстве случаев, временные таблицы создаются не вручную, а при использовании внешних данных для запроса, или при распределённом (`GLOBAL`) `IN`. Подробнее см. соответствующие разделы

Вместо временных можно использовать обычные таблицы с `ENGINE = Memory`.

## REPLACE TABLE

Запрос `REPLACE` позволяет частично изменить таблицу (строку или данные).

### Замечание

Такие запросы поддерживаются только движком БД **Atomic**.

Чтобы удалить часть данных из таблицы, вы можете создать новую таблицу, добавить в нее данные из старой таблицы, которые вы хотите оставить (отобрав их с помощью запроса `SELECT`), затем удалить старую таблицу и переименовать новую таблицу так как старую:

```
CREATE TABLE myNewTable AS myOldTable;
INSERT INTO myNewTable SELECT * FROM myOldTable WHERE CounterID <12345;
DROP TABLE myOldTable;
RENAME TABLE myNewTable TO myOldTable;
```

Вместо перечисленных выше операций можно использовать один запрос:

```
REPLACE TABLE myOldTable SELECT * FROM myOldTable WHERE CounterID <12345;
```

## Синтаксис

```
{CREATE [OR REPLACE]|REPLACE} TABLE [db.]table_name
```

Для данного запроса можно использовать любые варианты синтаксиса запроса `CREATE`. Запрос `REPLACE` для несуществующей таблицы вызовет ошибку.

## Примеры:

Рассмотрим таблицу:

```
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

Используем запрос `REPLACE` для удаления всех данных:

```
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;
```

n	s
2	\N

Используем запрос `REPLACE` для изменения структуры таблицы:

```
REPLACE TABLE base.t1 (n UInt64) ENGINE = MergeTree ORDER BY n;
INSERT INTO base.t1 VALUES (3);
SELECT * FROM base.t1;
```

n
3

## Секция COMMENT

Вы можете добавить комментарий к таблице при ее создании.

### Замечание

Комментарий поддерживается для всех движков таблиц, кроме **Kafka**, **RabbitMQ** и **EmbeddedRocksDB**.

### Синтаксис

```
CREATE TABLE db.table_name
(
    name1 type1, name2 type2, ...
)
ENGINE = engine
COMMENT 'Comment'
```

### Пример

Запрос:

```
CREATE TABLE t1 (x String) ENGINE = Memory COMMENT 'The temporary table';
SELECT name, comment FROM system.tables WHERE name = 't1';
```

Результат:

name	comment
t1	The temporary table

## CREATE VIEW

Создаёт представление. Представления бывают **обычные**, **материализованные** (MATERIALIZED) и **LIVE**.

### Обычные представления

```
CREATE [OR REPLACE] VIEW [IF NOT EXISTS] [db.]table_name [ON CLUSTER] AS SELECT ...
```

Обычные представления не хранят никаких данных, они выполняют чтение данных из другой таблицы при каждом доступе. Другими словами, обычное представление — это не что иное, как сохраненный запрос. При чтении данных из представления этот сохраненный запрос используется как подзапрос в секции **FROM**.

Для примера, пусть вы создали представление:

```
CREATE VIEW view AS SELECT ...
```

и написали запрос:

```
SELECT a, b, c FROM view
```

Этот запрос полностью эквивалентен использованию подзапроса:

```
SELECT a, b, c FROM (SELECT ...)
```

### Материализованные представления

```
CREATE MATERIALIZED VIEW [IF NOT EXISTS] [db.]table_name [ON CLUSTER] [TO[db.]name] [ENGINE = engine] [POPULATE] AS SELECT ...
```

Материализованные (MATERIALIZED) представления хранят данные, преобразованные соответствующим запросом **SELECT**.

При создании материализованного представления без использования **TO [db].[table]**, нужно обязательно указать **ENGINE** - движок таблицы для хранения данных.

При создании материализованного представления с использованием **TO [db].[table]**, нельзя указывать **POPULATE**.

Материализованное представление устроено следующим образом: при вставке данных в таблицу, указанную в **SELECT**-е, кусок вставляемых данных преобразуется этим запросом **SELECT**, и полученный результат вставляется в представление.

**Важно**

Материализованные представления в ClickHouse используют **имена столбцов** вместо порядка следования столбцов при вставке в целевую таблицу. Если в результатах запроса `SELECT` некоторые имена столбцов отсутствуют, то ClickHouse использует значение по умолчанию, даже если столбец не является **Nullable**. Безопасной практикой при использовании материализованных представлений считается добавление псевдонимов для каждого столбца.

Материализованные представления в ClickHouse больше похожи на `after insert` триггеры. Если в запросе материализованного представления есть агрегирование, оно применяется только к вставляемому блоку записей. Любые изменения существующих данных исходной таблицы (например обновление, удаление, удаление раздела и т.д.) не изменяют материализованное представление.

Если указано `POPULATE`, то при создании представления в него будут добавлены данные, уже содержащиеся в исходной таблице, как если бы был сделан запрос `CREATE TABLE ... AS SELECT ...`. Если `POPULATE` не указано, представление будет содержать только данные, добавленные в таблицу после создания представления. Использовать `POPULATE` не рекомендуется, так как в представление не попадут данные, добавляемые в таблицу во время создания представления.

Запрос `SELECT` может содержать `DISTINCT`, `GROUP BY`, `ORDER BY`, `LIMIT...` Следует иметь ввиду, что соответствующие преобразования будут выполняться независимо, на каждый блок вставляемых данных. Например, при наличии `GROUP BY`, данные будут агрегироваться при вставке, но только в рамках одной пачки вставляемых данных. Далее, данные не будут доагрегированы. Исключение - использование `ENGINE`, производящего агрегацию данных самостоятельно, например, `SummingMergeTree`.

Выполнение запросов `ALTER` над материализованными представлениями имеет свои особенности, поэтому эти запросы могут быть неудобными для использования. Если материализованное представление использует конструкцию `TO [db.]name`, то можно выполнить `DETACH` представления, `ALTER` для целевой таблицы и последующий `ATTACH` ранее отсоединенного (`DETACH`) представления.

Обратите внимание, что работа материализованного представления находится под влиянием настройки `optimize_on_insert`. Перед вставкой данных в таблицу происходит их слияние.

Представления выглядят так же, как обычные таблицы. Например, они перечисляются в результате запроса `SHOW TABLES`.

Чтобы удалить представление, следует использовать `DROP VIEW`. Впрочем, `DROP TABLE` тоже работает для представлений.

## LIVE-представления [экспериментальный функционал]

### Важно

Представления `LIVE VIEW` являются экспериментальной возможностью. Их использование может повлечь потерю совместимости в будущих версиях.

Чтобы использовать `LIVE VIEW` и запросы `WATCH`, включите настройку `allow_experimental_live_view`.

```
CREATE LIVE VIEW [IF NOT EXISTS] [db.]table_name [WITH [TIMEOUT [value_in_sec] [AND]]] [REFRESH [value_in_sec]]]
AS SELECT ...
```

LIVE VIEW хранит результат запроса **SELECT**, указанного при создании, и обновляется сразу же при изменении этого результата. Конечный результат запроса и промежуточные данные, из которых формируется результат, хранятся в оперативной памяти, и это обеспечивает высокую скорость обработки для повторяющихся запросов. LIVE-представления могут отправлять push-уведомления при изменении результата исходного запроса **SELECT**. Для этого используйте запрос **WATCH**.

Изменение **LIVE VIEW** запускается при вставке данных в таблицу, указанную в исходном запросе **SELECT**.

LIVE-представления работают по тому же принципу, что и распределенные таблицы. Но вместо объединения отдельных частей данных с разных серверов, LIVE-представления объединяют уже имеющийся результат с новыми данными. Если в исходном запросе LIVE-представления есть вложенный подзапрос, его результаты не кешируются, в кеше хранится только результат основного запроса.

## Ограничения

- **Табличные функции** в основном запросе не поддерживаются.
- Таблицы, не поддерживающие изменение с помощью запроса **INSERT**, такие как **словари** и **системные таблицы**, а также **нормальные представления** или **материализованные представления**, не запускают обновление LIVE-представления.
- В LIVE-представлениях могут использоваться только такие запросы, которые объединяют результаты по старым и новым данным. LIVE-представления не работают с запросами, требующими полного пересчета данных или агрегирования с сохранением состояния.
- LIVE VIEW не работает для реплицируемых и распределенных таблиц, добавление данных в которые происходит на разных узлах.
- LIVE VIEW не обновляется, если в исходном запросе используются несколько таблиц.

В случаях, когда **LIVE VIEW** не обновляется автоматически, чтобы обновлять его принудительно с заданной периодичностью, используйте **WITH REFRESH**.

## Отслеживание изменений LIVE-представлений

Для отслеживания изменений LIVE-представления используйте запрос **WATCH**.

### Пример:

```
CREATE TABLE mt (x Int8) Engine = MergeTree ORDER BY x;
CREATE LIVE VIEW lv AS SELECT sum(x) FROM mt;
```

Отслеживаем изменения LIVE-представления при вставке данных в исходную таблицу.

```
WATCH lv;
```

sum(x)	version
1	1
3	2
6	3

```
INSERT INTO mt VALUES (1);
INSERT INTO mt VALUES (2);
INSERT INTO mt VALUES (3);
```

Для получения списка изменений используйте ключевое слово **EVENTS**.

```
WATCH lv EVENTS;
```

version
1
2
3
...

Для работы с LIVE-представлениями, как и с любыми другими, можно использовать запросы **SELECT**. Если результат запроса кеширован, он будет возвращен немедленно, без обращения к исходным таблицам представления.

```
SELECT * FROM [db.]live_view WHERE ...
```

## Принудительное обновление LIVE-представлений

Чтобы принудительно обновить LIVE-представление, используйте запрос `ALTER LIVE VIEW [db.]table_name REFRESH`.

## Секция WITH TIMEOUT

LIVE-представление, созданное с параметром `WITH TIMEOUT`, будет автоматически удалено через определенное количество секунд с момента предыдущего запроса **WATCH**, примененного к данному LIVE-представлению.

```
CREATE LIVE VIEW [db.]table_name WITH TIMEOUT [value_in_sec] AS SELECT ...
```

Если временной промежуток не указан, используется значение настройки `temporary_live_view_timeout`.

### Пример:

```
CREATE TABLE mt (x Int8) Engine = MergeTree ORDER BY x;
CREATE LIVE VIEW lv WITH TIMEOUT 15 AS SELECT sum(x) FROM mt;
```

## Секция WITH REFRESH

LIVE-представление, созданное с параметром `WITH REFRESH`, будет автоматически обновляться через указанные промежутки времени, начиная с момента последнего обновления.

```
CREATE LIVE VIEW [db.]table_name WITH REFRESH [value_in_sec] AS SELECT ...
```

Если значение временного промежутка не задано, используется значение `periodic_live_view_refresh`.

### Пример:

```
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

Параметры `WITH TIMEOUT` и `WITH REFRESH` можно сочетать с помощью `AND`.

```
CREATE LIVE VIEW [db.]table_name WITH TIMEOUT [value_in_sec] AND REFRESH [value_in_sec] AS SELECT ...
```

### Пример:

```
CREATE LIVE VIEW lv WITH TIMEOUT 15 AND REFRESH 5 AS SELECT now();
```

По истечении 15 секунд представление будет автоматически удалено, если нет активного запроса `WATCH`.

```
WATCH lv;
```

```
Code: 60. DB::Exception: Received from localhost:9000. DB::Exception: Table default.lv doesn't exist..
```

## Использование LIVE-представлений

Наиболее частые случаи использования LIVE-представлений:

- Получение push-уведомлений об изменениях данных без дополнительных периодических запросов.
- Кэширование результатов часто используемых запросов для получения их без задержки.
- Отслеживание изменений таблицы для запуска других запросов `SELECT`.
- Отслеживание показателей из системных таблиц с помощью периодических обновлений.

## CREATE FUNCTION

Создает пользовательскую функцию из лямбда-выражения. Выражение должно состоять из параметров функции, констант, операторов и вызовов других функций.

## Синтаксис

```
CREATE FUNCTION name AS (parameter0, ...) -> expression
```

У функции может быть произвольное число параметров.

Существует несколько ограничений на создаваемые функции:

- Имя функции должно быть уникальным среди всех пользовательских и системных функций.
- Рекурсивные функции запрещены.
- Все переменные, используемые функцией, должны быть перечислены в списке ее параметров.

Если какое-нибудь ограничение нарушается, то при попытке создать функцию возникает исключение.

## Пример

Запрос:

```
CREATE FUNCTION linear_equation AS (x, k, b) -> k*x + b;  
SELECT number, linear_equation(number, 2, 1) FROM numbers(3);
```

Результат:

number	plus(multiply(2, number), 1)
0	1
1	3
2	5

В следующем запросе пользовательская функция вызывает **условную функцию**:

```
CREATE FUNCTION parity_str AS (n) -> if(n % 2, 'odd', 'even');  
SELECT number, parity_str(number) FROM numbers(3);
```

Результат:

number	if(modulo(number, 2), 'odd', 'even')
0	even
1	odd
2	even

# CREATE DICTIONARY

```
CREATE DICTIONARY [IF NOT EXISTS] [db.]dictionary_name [ON CLUSTER cluster]  
(  
    key1 type1 [DEFAULT|EXPRESSION expr1] [HIERARCHICAL|INJECTIVE|IS_OBJECT_ID],  
    key2 type2 [DEFAULT|EXPRESSION expr2] [HIERARCHICAL|INJECTIVE|IS_OBJECT_ID],  
    attr1 type2 [DEFAULT|EXPRESSION expr3],  
    attr2 type2 [DEFAULT|EXPRESSION expr4]  
)  
PRIMARY KEY key1, key2  
SOURCE(SOURCE_NAME([param1 value1 ... paramN valueN]))  
LAYOUT(LAYOUT_NAME([param_name param_value]))  
LIFETIME({MIN min_val MAX max_val | max_val})
```

Создаёт внешний словарь с заданной [структурой](#), [источником](#), способом размещения в памяти и периодом обновления.

Структура внешнего словаря состоит из атрибутов. Атрибуты словаря задаются как столбцы таблицы. Единственным обязательным свойством атрибута является его тип, все остальные свойства могут иметь значения по умолчанию.

В зависимости от [способа размещения словаря в памяти](#), ключами словаря могут быть один и более атрибутов.

Смотрите [Внешние словари](#).

## CREATE USER

Создает [аккаунты пользователей](#).

Синтаксис:

```
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 Ildap 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 позволяет создавать пользователей в кластере, см. [Распределенные DDL](#).

## Идентификация

Существует несколько способов идентификации пользователя:

- 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 Ildap SERVER 'server\_name'
- IDENTIFIED WITH kerberos or IDENTIFIED WITH kerberos REALM 'realm'

## Пользовательский хост

Пользовательский хост — это хост, с которого можно установить соединение с сервером ClickHouse. Хост задается в секции `HOST` следующими способами:

- HOST IP 'ip\_address\_or\_subnetwork' — Пользователь может подключиться к серверу ClickHouse только с указанного IP-адреса или [подсети](#). Примеры: `HOST IP '192.168.0.0/16'`, `HOST IP '2001:DB8::/32'`. При использовании в эксплуатации указывайте только элементы `HOST IP` (IP-адреса и маски подсети), так как использование `host` и `host_regexp` может привести к дополнительной задержке.

- `HOST ANY` — Пользователь может подключиться с любого хоста. Используется по умолчанию.
- `HOST LOCAL` — Пользователь может подключиться только локально.
- `HOST NAME 'fqdn'` — Хост задается через FQDN. Например, `HOST NAME 'mysite.com'`.
- `HOST NAME REGEXP 'regexp'` — Позволяет использовать регулярные выражения `pcre`, чтобы задать хосты. Например, `HOST NAME REGEXP '.*\.mysite\.com'`.
- `HOST LIKE 'template'` — Позволяет использовать оператор `LIKE` для фильтрации хостов. Например, `HOST LIKE '%'` эквивалентен `HOST ANY`; `HOST LIKE '%.mysite.com'` разрешает подключение со всех хостов в домене `mysite.com`.

Также, чтобы задать хост, вы можете использовать `@` вместе с именем пользователя. Примеры:

- `CREATE USER mira@'127.0.0.1'` — Эквивалентно `HOST IP`.
- `CREATE USER mira@'localhost'` — Эквивалентно `HOST LOCAL`.
- `CREATE USER mira@'192.168.%.%'` — Эквивалентно `HOST LIKE`.

## Внимание

ClickHouse трактует конструкцию `user_name@'address'` как имя пользователя целиком. То есть технически вы можете создать несколько пользователей с одинаковыми `user_name`, но разными частями конструкции после `@`, но лучше так не делать.

## Секция GRANTEES

Указываются пользователи или роли, которым разрешено получать `привилегии` от создаваемого пользователя при условии, что этому пользователю также предоставлен весь необходимый доступ с использованием `GRANT OPTION`. Параметры секции `GRANTEES`:

- `user` — указывается пользователь, которому разрешено получать привилегии от создаваемого пользователя.
- `role` — указывается роль, которой разрешено получать привилегии от создаваемого пользователя.
- `ANY` — любому пользователю или любой роли разрешено получать привилегии от создаваемого пользователя. Используется по умолчанию.
- `NONE` — никому не разрешено получать привилегии от создаваемого пользователя.

Вы можете исключить любого пользователя или роль, используя выражение `EXCEPT`. Например, `CREATE USER user1 GRANTEES ANY EXCEPT user2`. Это означает, что если `user1` имеет привилегии, предоставленные с использованием `GRANT OPTION`, он сможет предоставить их любому, кроме `user2`.

## Примеры

Создать аккаунт `mira`, защищенный паролем `qwerty`:

```
CREATE USER mira HOST IP '127.0.0.1' IDENTIFIED WITH sha256_password BY 'qwerty';
```

Пользователь `mira` должен запустить клиентское приложение на хосте, где запущен ClickHouse.

Создать аккаунт `john`, назначить на него роли, сделать данные роли ролями по умолчанию:

```
CREATE USER john DEFAULT ROLE role1, role2;
```

Создать аккаунт `john` и установить ролями по умолчанию все его будущие роли:

```
CREATE USER john DEFAULT ROLE ALL;
```

Когда роль будет назначена аккаунту `john`, она автоматически станет ролью по умолчанию.

Создать аккаунт `john` и установить ролями по умолчанию все его будущие роли, кроме `role1` и `role2`:

```
CREATE USER john DEFAULT ROLE ALL EXCEPT role1, role2;
```

Создать пользователя с аккаунтом `john` и разрешить ему предоставить свои привилегии пользователю с аккаунтом `jack`:

```
CREATE USER john GRANTEEES jack;
```

## CREATE ROLE

Создает **роли**. Роль — это набор **привилегий**. Пользователь, которому назначена роль, получает все привилегии этой роли.

Синтаксис:

```
CREATE ROLE [IF NOT EXISTS | OR REPLACE] name1 [, name2 ...]
  [SETTINGS variable [= value] [MIN [=] min_value] [MAX [=] max_value] [READONLY|WRITABLE] | PROFILE
  'profile_name'] [...]
```

## Управление ролями

Одному пользователю можно назначить несколько ролей. Пользователи могут применять назначенные роли в произвольных комбинациях с помощью выражения **SET ROLE**. Конечный объем привилегий — это комбинация всех привилегий всех примененных ролей. Если у пользователя имеются привилегии, присвоенные его аккаунту напрямую, они также прибавляются к привилегиям, присвоенным через роли.

Роли по умолчанию применяются при входе пользователя в систему. Установить роли по умолчанию можно с помощью выражений **SET DEFAULT ROLE** или **ALTER USER**.

Для отзыва роли используется выражение **REVOKE**.

Для удаления роли используется выражение **DROP ROLE**. Удаленная роль автоматически отзывается у всех пользователей, которым была назначена.

## Примеры

```
CREATE ROLE accountant;
GRANT SELECT ON db.* TO accountant;
```

Такая последовательность запросов создаст роль `accountant`, у которой есть привилегия на чтение из базы данных `accounting`.

Назначить роль `accountant` аккаунту `mira`:

```
GRANT accountant TO mira;
```

После назначения роли пользователь может ее применить и выполнять разрешенные ей запросы.  
Например:

```
SET ROLE accountant;  
SELECT * FROM db.*;
```

## CREATE ROW POLICY

Создает **политики доступа к строкам**, т.е. фильтры, которые определяют, какие строки пользователь может читать из таблицы.

Синтаксис:

```
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 ...]  
[AS {PERMISSIVE | RESTRICTIVE}]  
[FOR SELECT] USING condition  
[TO {role [...] | ALL | ALL EXCEPT role [...]}]
```

## Секция USING

Секция **USING** указывает условие для фильтрации строк. Пользователь может видеть строку, если это условие, вычисленное для строки, дает ненулевой результат.

## Секция TO

В секции **TO** перечисляются пользователи и роли, для которых должна действовать политика.  
Например, **CREATE ROW POLICY ... TO accountant, john@localhost**.

Ключевым словом **ALL** обозначаются все пользователи, включая текущего. Ключевые слова **ALL EXCEPT** позволяют исключить пользователей из списка всех пользователей. Например, **CREATE ROW POLICY ... TO ALL EXCEPT accountant, john@localhost**

## Note

Если для таблицы не задано ни одной политики доступа к строкам, то любой пользователь может выполнить команду **SELECT** и получить все строки таблицы. Если определить хотя бы одну политику для таблицы, то доступ к строкам будет управляться этими политиками, причем для всех пользователей (даже для тех, для кого политики не определялись).

Например, следующая политика

```
CREATE ROW POLICY pol1 ON mydb.table1 USING b=1 TO mira, peter
```

запретит пользователям **mira** и **peter** видеть строки с **b != 1**, и еще запретит всем остальным пользователям (например, пользователю **paul**) видеть какие-либо строки вообще из таблицы **mydb.table1**.

Если это нежелательно, такое поведение можно исправить, определив дополнительную политику:

```
CREATE ROW POLICY pol2 ON mydb.table1 USING 1 TO ALL EXCEPT mira, peter
```

## Секция AS

Может быть одновременно активно более одной политики для одной и той же таблицы и одного и того же пользователя. Поэтому нам нужен способ комбинировать политики.

По умолчанию политики комбинируются с использованием логического оператора `OR`. Например, политики:

```
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
```

разрешат пользователю с именем `peter` видеть строки, для которых будет верно `b=1` или `c=2`.

Секция `AS` указывает, как политики должны комбинироваться с другими политиками. Политики могут быть или разрешительными (`PERMISSIVE`), или ограничительными (`RESTRICTIVE`). По умолчанию политики создаются разрешительными (`PERMISSIVE`); такие политики комбинируются с использованием логического оператора `OR`.

Ограничительные (`RESTRICTIVE`) политики комбинируются с использованием логического оператора `AND`.

Общая формула выглядит так:

```
строка_видима = (одна или больше permissive-политик дала ненулевой результат проверки условия) И  
(все restrictive-политики дали ненулевой результат проверки условия)
```

Например, политики

```
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
```

разрешат пользователю с именем `peter` видеть только те строки, для которых будет одновременно `b=1` и `c=2`.

## Секция ON CLUSTER

Секция `ON CLUSTER` позволяет создавать политики на кластере, см. [Распределенные DDL запросы](#).

## Примеры

```
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

Создает `квоту`, которая может быть присвоена пользователю или роли.

Синтаксис:

```
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 [...]}]
```

Ключи `user_name`, `ip_address`, `client_key`, `client_key, user_name` и `client_key, ip_address` соответствуют полям таблицы `system.quotas`.

Параметры `queries`, `query_selects`, `query_inserts`, `errors`, `result_rows`, `result_bytes`, `read_rows`, `read_bytes`, `execution_time` соответствуют полям таблицы `system.quotas_usage`.

В секции `ON CLUSTER` можно указать кластеры, на которых создается квота, см. [Распределенные DDL запросы](#).

## Примеры

Ограничить максимальное количество запросов для текущего пользователя — не более 123 запросов за каждые 15 месяцев:

```
CREATE QUOTA qA FOR INTERVAL 15 month MAX queries = 123 TO CURRENT_USER;
```

Ограничить по умолчанию максимальное время выполнения запроса — не более полсекунды за каждые 30 минут, а также максимальное число запросов — не более 321 и максимальное число ошибок — не более 10 за каждые 5 кварталов:

```
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

Создает [профили настроек](#), которые могут быть присвоены пользователю или роли.

Синтаксис:

```
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'] [...]
```

Секция `ON CLUSTER` позволяет создавать профили на кластере, см. [Распределенные DDL запросы](#).

## Пример

Создать профиль настроек `max_memory_usage_profile`, который содержит значение и ограничения для настройки `max_memory_usage`. Присвоить профиль пользователю `robin`:

```
CREATE SETTINGS PROFILE max_memory_usage_profile SETTINGS max_memory_usage = 100000001 MIN 90000000
MAX 110000000 TO robin
```

# ALTER

Изменение структуры таблицы.

```
ALTER TABLE [db].name [ON CLUSTER cluster] ADD|DROP|CLEAR|COMMENT|MODIFY COLUMN ...
```

В запросе указывается список из одного или более действий через запятую. Каждое действие — операция над столбцом.

Большинство запросов `ALTER TABLE` изменяют настройки таблицы или данные:

- `COLUMN`
- `PARTITION`
- `DELETE`
- `UPDATE`
- `ORDER BY`
- `INDEX`
- `CONSTRAINT`
- `TTL`

## Note

Запрос `ALTER TABLE` поддерживается только для таблиц типа `*MergeTree`, а также `Merge` и `Distributed`. Запрос имеет несколько вариантов.

Следующие запросы `ALTER` управляют представлениями:

- `ALTER TABLE ... MODIFY QUERY` — изменяет структуру `Materialized view`.
- `ALTER LIVE VIEW` — обновляет `Live view`.

Следующие запросы `ALTER` изменяют сущности, связанные с управлением доступом на основе ролей:

- `USER`
- `ROLE`
- `QUOTA`
- `ROW POLICY`
- `SETTINGS PROFILE`

## Мутации

Мутации - разновидность запроса `ALTER`, позволяющая изменять или удалять данные в таблице. В отличие от стандартных запросов `ALTER TABLE ... DELETE` и `ALTER TABLE ... UPDATE`, рассчитанных на точечное изменение данных, область применения мутаций - достаточно тяжёлые изменения, затрагивающие много строк в таблице. Поддержана для движков таблиц семейства `MergeTree`, в том числе для движков с репликацией.

Конвертировать существующие таблицы для работы с мутациями не нужно. Но после применения первой мутации формат данных таблицы становится несовместимым с предыдущими версиями и откатиться на предыдущую версию уже не получится.

На данный момент доступны команды:

```
ALTER TABLE [db.]table MATERIALIZE INDEX name IN PARTITION partition_name
```

Команда перестроит вторичный индекс name для партиции partition\_name.

В одном запросе можно указать несколько команд через запятую.

Для \*MergeTree-таблиц мутации выполняются, перезаписывая данные по кускам (parts). При этом атомарности нет — куски заменяются на помутуированные по мере выполнения и запрос SELECT, заданный во время выполнения мутации, увидит данные как из измененных кусков, так и из кусков, которые еще не были изменены.

Мутации линейно упорядочены между собой и накладываются на каждый кусок в порядке добавления. Мутации также упорядочены со вставками - гарантируется, что данные, вставленные в таблицу до начала выполнения запроса мутации, будут изменены, а данные, вставленные после окончания запроса мутации, изменены не будут. При этом мутации никак не блокируют вставки.

Запрос завершается немедленно после добавления информации о мутации (для реплицированных таблиц - в ZooKeeper, для нереплицированных - на файловую систему). Сама мутация выполняется асинхронно, используя настройки системного профиля. Следить за ходом её выполнения можно по таблице system.mutations. Добавленные мутации будут выполняться до конца даже в случае перезапуска серверов ClickHouse. Откатить мутацию после её добавления нельзя, но если мутация по какой-то причине не может выполниться до конца, её можно остановить с помощью запроса KILL MUTATION.

Записи о последних выполненных мутациях удаляются не сразу (количество сохраняемых мутаций определяется параметром движка таблиц finished\_mutations\_to\_keep). Более старые записи удаляются.

## Синхронность запросов ALTER

Для нереплицируемых таблиц, все запросы ALTER выполняются синхронно. Для реплицируемых таблиц, запрос всего лишь добавляет инструкцию по соответствующим действиям в ZooKeeper, а сами действия осуществляются при первой возможности. Но при этом, запрос может ждать завершения выполнения этих действий на всех репликах.

Для всех запросов ALTER можно настроить ожидание с помощью настройки `replication_alter_partitions_sync`.

Вы можете указать время ожидания (в секундах) выполнения всех запросов ALTER для неактивных реплик с помощью настройки `replication_wait_for_inactive_replica_timeout`.

### Примечание

Для всех запросов ALTER при `replication_alter_partitions_sync = 2` и неактивности некоторых реплик больше времени, заданного настройкой `replication_wait_for_inactive_replica_timeout`, генерируется исключение `UNFINISHED`.

Для запросов `ALTER TABLE ... UPDATE|DELETE` синхронность выполнения определяется настройкой `mutations_sync`.

## Манипуляции со столбцами

Набор действий, позволяющих изменять структуру таблицы.

Синтаксис:

```
ALTER TABLE [db].name [ON CLUSTER cluster] ADD|DROP|RENAME|CLEAR|COMMENT|MODIFY|MATERIALIZE COLUMN  
...
```

В запросе можно указать сразу несколько действий над одной таблицей через запятую. Каждое действие — это манипуляция над столбцом.

Существуют следующие действия:

- **ADD COLUMN** — добавляет столбец в таблицу;
- **DROP COLUMN** — удаляет столбец;
- **RENAME COLUMN** — переименовывает существующий столбец;
- **CLEAR COLUMN** — сбрасывает все значения в столбце для заданной партиции;
- **COMMENT COLUMN** — добавляет комментарий к столбцу;
- **MODIFY COLUMN** — изменяет тип столбца, выражение для значения по умолчанию и TTL;
- **MODIFY COLUMN REMOVE** — удаляет какое-либо из свойств столбца;
- **MATERIALIZE COLUMN** — делает столбец материализованным (**MATERIALIZED**) в кусках, в которых отсутствуют значения.

Подробное описание для каждого действия приведено ниже.

## ADD COLUMN

```
ADD COLUMN [IF NOT EXISTS] name [type] [default_expr] [codec] [AFTER name_after | FIRST]
```

Добавляет в таблицу новый столбец с именем `name`, типом `type`, **кодеком** `codec` и выражением для умолчания `default_expr` (смотрите раздел [Значения по умолчанию](#)).

Если указано `IF NOT EXISTS`, запрос не будет возвращать ошибку, если столбец уже существует. Если указано `AFTER name_after` (имя другого столбца), то столбец добавляется (в список столбцов таблицы) после указанного. Если вы хотите добавить столбец в начало таблицы, используйте `FIRST`. Иначе столбец добавляется в конец таблицы. Для цепочки действий `name_after` может быть именем столбца, который добавляется в одном из предыдущих действий.

Добавление столбца всего лишь меняет структуру таблицы, и не производит никаких действий с данными - соответствующие данные не появляются на диске после ALTER-а. При чтении из таблицы, если для какого-либо столбца отсутствуют данные, то он заполняется значениями по умолчанию (выполняя выражение по умолчанию, если такое есть, или нулями, пустыми строками). Также, столбец появляется на диске при слиянии кусков данных (см. [MergeTree](#)).

Такая схема позволяет добиться мгновенной работы запроса ALTER и отсутствия необходимости увеличивать объём старых данных.

Пример:

```
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
```

Удаляет столбец с именем `name`. Если указано `IF EXISTS`, запрос не будет возвращать ошибку, если столбца не существует.

Запрос удаляет данные из файловой системы. Так как это представляет собой удаление целых файлов, запрос выполняется почти мгновенно.

### Предупреждение

Вы не можете удалить столбец, используемый в **материализованном представлении**. В противном случае будет ошибка.

Пример:

```
ALTER TABLE visits DROP COLUMN browser
```

## RENAME COLUMN

```
RENAME COLUMN [IF EXISTS] name to new_name
```

Переименовывает столбец `name` в `new_name`. Если указано выражение `IF EXISTS`, то запрос не будет возвращать ошибку при условии, что столбец `name` не существует. Поскольку переименование не затрагивает физические данные колонки, запрос выполняется практически мгновенно.

**ЗАМЕЧЕНИЕ:** Столбцы, являющиеся частью основного ключа или ключа сортировки (заданные с помощью `ORDER BY` или `PRIMARY KEY`), не могут быть переименованы. Попытка переименовать эти слобцы приведет к `SQL Error [524]`.

Пример:

```
ALTER TABLE visits RENAME COLUMN webBrowser TO browser
```

## CLEAR COLUMN

```
CLEAR COLUMN [IF EXISTS] name IN PARTITION partition_name
```

Сбрасывает все значения в столбце для заданной партиции. Если указано `IF EXISTS`, запрос не будет возвращать ошибку, если столбца не существует.

Как корректно задать имя партиции, см. в разделе [Как задавать имя партиции в запросах ALTER](#).

Пример:

```
ALTER TABLE visits CLEAR COLUMN browser IN PARTITION tuple()
```

## COMMENT COLUMN

```
COMMENT COLUMN [IF EXISTS] name 'Text comment'
```

Добавляет комментарий к таблице. Если указано IF EXISTS, запрос не будет возвращать ошибку, если столбца не существует.

Каждый столбец может содержать только один комментарий. При выполнении запроса существующий комментарий заменяется на новый.

Посмотреть комментарии можно в столбце `comment_expression` из запроса [DESCRIBE TABLE](#).

Пример:

```
ALTER TABLE visits COMMENT COLUMN browser 'Столбец показывает, из каких браузеров пользователи заходили на сайт.'
```

## MODIFY COLUMN

```
MODIFY COLUMN [IF EXISTS] name [type] [default_expr] [codec] [TTL] [AFTER name_after | FIRST]
```

Запрос изменяет следующие свойства столбца `name`:

- Тип
- Значение по умолчанию
- Кодеки сжатия
- TTL

Примеры изменения кодеков сжатия смотрите в разделе [Кодеки сжатия столбцов](#).

Примеры изменения TTL столбца смотрите в разделе [TTL столбца](#).

Если указано IF EXISTS, запрос не возвращает ошибку при условии, что столбец не существует.

Запрос также может изменять порядок столбцов при помощи FIRST | AFTER, смотрите описание [ADD COLUMN](#).

При изменении типа, значения преобразуются так, как если бы к ним была применена функция [toType](#). Если изменяется только выражение для умолчания, запрос не делает никакой сложной работы и выполняется мгновенно.

Пример запроса:

```
ALTER TABLE visits MODIFY COLUMN browser Array(String)
```

Изменение типа столбца - это единственное действие, которое выполняет сложную работу - меняет содержимое файлов с данными. Для больших таблиц, выполнение может занять длительное время.

Выполнение запроса ALTER атомарно.

Запрос ALTER на изменение столбцов реплицируется. Соответствующие инструкции сохраняются в ZooKeeper, и затем каждая реплика их применяет. Все запросы ALTER выполняются в одном и том же порядке. Запрос ждёт выполнения соответствующих действий на всех репликах. Но при этом, запрос на изменение столбцов в реплицируемой таблице можно прервать, и все действия будут осуществлены асинхронно.

## MODIFY COLUMN REMOVE

Удаляет какое-либо из свойств столбца: DEFAULT, ALIAS, MATERIALIZED, CODEC, COMMENT, TTL.

Синтаксис:

```
ALTER TABLE table_name MODIFY column_name REMOVE property;
```

### Пример

Удаление свойства TTL:

```
ALTER TABLE table_with_ttl MODIFY COLUMN column_ttl REMOVE TTL;
```

### Смотрите также

- [REMOVE TTL](#).

## MATERIALIZE COLUMN

Материализует столбец таблицы в кусках, в которых отсутствуют значения. Используется, если необходимо создать новый столбец со сложным материализованным выражением или выражением для заполнения по умолчанию (DEFAULT), потому как вычисление такого столбца прямо во время выполнения запроса SELECT оказывается ощутимо затратным. Чтобы совершить ту же операцию для существующего столбца, используйте модификатор FINAL.

Синтаксис:

```
ALTER TABLE table MATERIALIZE COLUMN col [FINAL];
```

### Пример

```
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;
```

### Результат:

```
groupArray(x)-----| groupArray(s)-----|
[0,1,2,3,4,5,6,7,8,9] | ['0','1','2','3','4','5','6','7','8','9'] |
```

## Ограничения запроса ALTER

Запрос `ALTER` позволяет создавать и удалять отдельные элементы (столбцы) вложенных структур данных, но не вложенные структуры данных целиком. Для добавления вложенной структуры данных, вы можете добавить столбцы с именем вида `name.nested_name` и типом `Array(T)` - вложенная структура данных полностью эквивалентна нескольким столбцам-массивам с именем, имеющим одинаковый префикс до точки.

Отсутствует возможность удалять столбцы, входящие в первичный ключ или ключ для сэмплирования (в общем, входящие в выражение `ENGINE`). Изменение типа у столбцов, входящих в первичный ключ возможно только в том случае, если это изменение не приводит к изменению данных (например, разрешено добавление значения в `Enum` или изменение типа с `DateTime` на `UInt32`).

Если возможностей запроса `ALTER` не хватает для нужного изменения таблицы, вы можете создать новую таблицу, скопировать туда данные с помощью запроса `INSERT SELECT`, затем поменять таблицы местами с помощью запроса `RENAME`, и удалить старую таблицу. В качестве альтернативы для запроса `INSERT SELECT`, можно использовать инструмент `clickhouse-copier`.

Запрос `ALTER` блокирует все чтения и записи для таблицы. То есть если на момент запроса `ALTER` выполнялся долгий `SELECT`, то запрос `ALTER` сначала дождётся его выполнения. И в это время все новые запросы к той же таблице будут ждать, пока завершится этот `ALTER`.

Для таблиц, которые не хранят данные самостоятельно (типа `Merge` и `Distributed`), `ALTER` всего лишь меняет структуру таблицы, но не меняет структуру подчинённых таблиц. Для примера, при `ALTER`-е таблицы типа `Distributed`, вам также потребуется выполнить запрос `ALTER` для таблиц на всех удалённых серверах.

## Манипуляции сパーティциями и кусками

Для работы с `партициями` доступны следующие операции:

- `DETACH PARTITION` — перенести партицию в директорию `detached`;
- `DROP PARTITION` — удалить партицию;
- `ATTACH PARTITION|PART` — добавить партицию/кусок в таблицу из директории `detached`;
- `ATTACH PARTITION FROM` — скопировать партицию из другой таблицы;
- `REPLACE PARTITION` — скопировать партицию из другой таблицы с заменой;
- `MOVE PARTITION TO TABLE` — переместить партицию в другую таблицу;
- `CLEAR COLUMN IN PARTITION` — удалить все значения в столбце для заданной партиции;
- `CLEAR INDEX IN PARTITION` — очистить построенные вторичные индексы для заданной партиции;
- `FREEZE PARTITION` — создать резервную копию партиции;
- `UNFREEZE PARTITION` — удалить резервную копию партиции;
- `FETCH PARTITION|PART` — скачать партицию/кусок с другого сервера;
- `MOVE PARTITION|PART` — переместить партицию/кусок на другой диск или том.
- `UPDATE IN PARTITION` — обновить данные внутри партиции по условию.
- `DELETE IN PARTITION` — удалить данные внутри партиции по условию.

### `DETACH PARTITION|PART`

```
ALTER TABLE table_name DETACH PARTITION|PART partition_expr
```

Перемещает заданную партицию в директорию `detached`. Сервер не будет знать об этой партиции до тех пор, пока вы не выполните запрос [ATTACH](#).

Пример:

```
ALTER TABLE mt DETACH PARTITION '2020-11-21';
ALTER TABLE mt DETACH PART 'all_2_2_0';
```

Подробнее о том, как корректно задать имя партиции, см. в разделе [Как задавать имя партиции в запросах ALTER](#).

После того как запрос будет выполнен, вы сможете производить любые операции с данными в директории `detached`. Например, можно удалить их из файловой системы.

Запрос реплицируется — данные будут перенесены в директорию `detached` и забыты на всех репликах. Обратите внимание, запрос может быть отправлен только на реплику-лидер. Чтобы узнать, является ли реплика лидером, выполните запрос `SELECT` к системной таблице `system.replicas`. Либо можно выполнить запрос `DETACH` на всех репликах — тогда на всех репликах, кроме реплик-лидеров (поскольку допускается несколько лидеров), запрос вернет ошибку.

## DROP PARTITION|PART

```
ALTER TABLE table_name DROP PARTITION|PART partition_expr
```

Удаляет партицию. Партиция помечается как неактивная и будет полностью удалена примерно через 10 минут.

Подробнее о том, как корректно задать имя партиции, см. в разделе [Как задавать имя партиции в запросах ALTER](#).

Запрос реплицируется — данные будут удалены на всех репликах.

Пример:

```
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
```

Удаляет из `detached` кусок или все куски, принадлежащие партиции.

Подробнее о том, как корректно задать имя партиции, см. в разделе [Как задавать имя партиции в запросах ALTER](#).

## ATTACH PARTITION|PART

```
ALTER TABLE table_name ATTACH PARTITION|PART partition_expr
```

Добавляет данные в таблицу из директории `detached`. Можно добавить данные как для целой партиции, так и для отдельного куска. Примеры:

```
ALTER TABLE visits ATTACH PARTITION 201901;
ALTER TABLE visits ATTACH PART 201901_2_2_0;
```

Как корректно задать имя партиции или куска, см. в разделе [Как задавать имя партиции в запросах ALTER](#).

Этот запрос реплицируется. Реплика-иницатор проверяет, есть ли данные в директории `detached`. Если данные есть, то запрос проверяет их целостность. В случае успеха данные добавляются в таблицу.

Если реплика, не являющаяся инициатором запроса, получив команду присоединения, находит кусок с правильными контрольными суммами в своей собственной папке `detached`, она присоединяет данные, не скачивая их с других реплик.

Если нет куска с правильными контрольными суммами, данные загружаются из любой реплики, имеющей этот кусок.

Вы можете поместить данные в директорию `detached` на одной реплике и с помощью запроса `ALTER ... ATTACH` добавить их в таблицу на всех репликах.

## ATTACH PARTITION FROM

```
ALTER TABLE table2 ATTACH PARTITION partition_expr FROM table1
```

Копирует партицию из таблицы `table1` в таблицу `table2`.

Обратите внимание, что данные не удаляются ни из `table1`, ни из `table2`.

Следует иметь в виду:

- Таблицы должны иметь одинаковую структуру.
- Для таблиц должен быть задан одинаковый ключ партиционирования.

Подробнее о том, как корректно задать имя партиции, см. в разделе [Как задавать имя партиции в запросах ALTER](#).

## REPLACE PARTITION

```
ALTER TABLE table2 REPLACE PARTITION partition_expr FROM table1
```

Копирует партицию из таблицы `table1` в таблицу `table2` с заменой существующих данных в `table2`.  
Данные из `table1` не удаляются.

Следует иметь в виду:

- Таблицы должны иметь одинаковую структуру.
- Для таблиц должен быть задан одинаковый ключ партиционирования.

Подробнее о том, как корректно задать имя партиции, см. в разделе [Как задавать имя партиции в запросах ALTER](#).

## MOVE PARTITION TO TABLE

```
ALTER TABLE table_source MOVE PARTITION partition_expr TO TABLE table_dest
```

Перемещает партицию из таблицы `table_source` в таблицу `table_dest` (добавляет к существующим данным в `table_dest`) с удалением данных из таблицы `table_source`.

Следует иметь в виду:

- Таблицы должны иметь одинаковую структуру.
- Для таблиц должен быть задан одинаковый ключpartitionирования.
- Движки таблиц должны быть одинакового семейства (реплицированные или нереплицированные).
- Для таблиц должна быть задана одинаковая политика хранения.

## CLEAR COLUMN IN PARTITION

```
ALTER TABLE table_name CLEAR COLUMN column_name IN PARTITION partition_expr
```

Сбрасывает все значения в столбце для заданной партиции. Если для столбца определено значение по умолчанию (в секции `DEFAULT`), то будет выставлено это значение.

Пример:

```
ALTER TABLE visits CLEAR COLUMN hour in PARTITION 201902
```

## CLEAR INDEX IN PARTITION

```
ALTER TABLE table_name CLEAR INDEX index_name IN PARTITION partition_expr
```

Работает как `CLEAR COLUMN`, но сбрасывает индексы вместо данных в столбцах.

## FREEZE PARTITION

```
ALTER TABLE table_name FREEZE [PARTITION partition_expr]
```

Создаёт резервную копию для заданной партиции. Если выражение `PARTITION` опущено, резервные копии будут созданы для всех партиций.

### Примечание

Создание резервной копии не требует остановки сервера.

Для таблиц старого стиля имя партиций можно задавать в виде префикса (например, `2019`). В этом случае, резервные копии будут созданы для всех соответствующих партиций. Подробнее о том, как корректно задать имя партиции, см. в разделе [Как задавать имя партиции в запросах ALTER](#).

Запрос формирует для текущего состояния таблицы жесткие ссылки на данные в этой таблице. Ссылки размещаются в директории `/var/lib/clickhouse/shadow/N/...`, где:

- `/var/lib/clickhouse/` — рабочая директория ClickHouse, заданная в конфигурационном файле;
- `N` — инкрементальный номер резервной копии.

## Примечание

При использовании **нескольких дисков для хранения данных таблицы** директория `shadow/N` появляется на каждом из дисков, на которых были куски, попавшие под выражение `PARTITION`.

Структура директорий внутри резервной копии такая же, как внутри `/var/lib/clickhouse/`. Запрос выполнит `chmod` для всех файлов, запрещая запись в них.

Обратите внимание, запрос `ALTER TABLE t FREEZE PARTITION` не реплицируется. Он создает резервную копию только на локальном сервере. После создания резервной копии данные из `/var/lib/clickhouse/shadow/` можно скопировать на удаленный сервер, а локальную копию удалить.

Резервная копия создается почти мгновенно (однако, сначала запрос дожидается завершения всех запросов, которые выполняются для соответствующей таблицы).

`ALTER TABLE t FREEZE PARTITION` копирует только данные, но не метаданные таблицы. Чтобы сделать резервную копию метаданных таблицы, скопируйте файл `/var/lib/clickhouse/metadata/database/table.sql`

Чтобы восстановить данные из резервной копии, выполните следующее:

- Создайте таблицу, если она ещё не существует. Запрос на создание можно взять из `.sql` файла (замените в нём `ATTACH` на `CREATE`).
- Скопируйте данные из директории `data/database/table/` внутри резервной копии в директорию `/var/lib/clickhouse/data/database/table/detached/`.
- С помощью запросов `ALTER TABLE t ATTACH PARTITION` добавьте данные в таблицу.

Восстановление данных из резервной копии не требует остановки сервера.

Подробнее о резервном копировании и восстановлении данных читайте в разделе [Резервное копирование данных](#).

## UNFREEZE PARTITION

```
ALTER TABLE 'table_name' UNFREEZE [PARTITION 'part_expr'] WITH NAME 'backup_name'
```

Удаляет с диска "замороженные" партиции с указанным именем. Если секция `PARTITION` опущена, запрос удаляет резервную копию всех партиций сразу.

## FETCH PARTITION|PART

```
ALTER TABLE table_name FETCH PARTITION|PART partition_expr FROM 'path-in-zookeeper'
```

Загружает партицию с другого сервера. Этот запрос работает только для реплицированных таблиц.

Запрос выполняет следующее:

- Загружает партицию/кусок с указанного шарда. Путь к шарду задается в секции `FROM` ('`path-in-zookeeper`'). Обратите внимание, нужно задавать путь к шарду в ZooKeeper.
- Помещает загруженные данные в директорию `detached` таблицы `table_name`. Чтобы прикрепить эти данные к таблице, используйте запрос [ATTACH PARTITION|PART](#).

Например:

## 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;
```

Следует иметь в виду:

- Запрос `ALTER TABLE t FETCH PARTITION|PART` не реплицируется. Он загружает партицию в директорию `detached` только на локальном сервере.
- Запрос `ALTER TABLE t ATTACH` реплицируется — он добавляет данные в таблицу сразу на всех репликах. На одной из реплик данные будут добавлены из директории `detached`, а на других — из соседних реплик.

Перед загрузкой данных система проверяет, существует ли партиция и совпадает ли её структура со структурой таблицы. При этом автоматически выбирается наиболее актуальная реплика среди всех живых реплик.

Несмотря на то что запрос называется `ALTER TABLE`, он не изменяет структуру таблицы и не изменяет сразу доступные данные в таблице.

## MOVE PARTITION|PART

Перемещает партицию или кусок данных на другой том или диск для таблиц с движком MergeTree. Смотрите [Хранение данных таблицы на нескольких блочных устройствах](#).

```
ALTER TABLE table_name MOVE PARTITION|PART partition_expr TO DISK|VOLUME 'disk_name'
```

Запрос `ALTER TABLE t MOVE`:

- Не реплицируется, т.к. на разных репликах могут быть различные конфигурации политик хранения.
- Возвращает ошибку, если указан несконфигурированный том или диск. Ошибка также возвращается в случае невыполнения условий перемещения данных, которые указаны в конфигурации политики хранения.
- Может возвращать ошибку в случае, когда перемещаемые данные уже оказались перемещены в результате фонового процесса, конкурентного запроса `ALTER TABLE t MOVE` или как часть результата фоновой операции слияния. В данном случае никаких дополнительных действий от пользователя не требуется.

Примеры:

```
ALTER TABLE hits MOVE PART '20190301_14343_16206_438' TO VOLUME 'slow'
ALTER TABLE hits MOVE PARTITION '2019-09-01' TO DISK 'fast_ssds'
```

## UPDATE IN PARTITION

Манипулирует данными в указанной партиции, соответствующими заданному выражению фильтрации. Реализовано как мутация [mutation](#).

Синтаксис:

```
ALTER TABLE [db.]table UPDATE column1 = expr1 [, ...] [IN PARTITION partition_id] WHERE filter_expr
```

## Пример

```
ALTER TABLE mt UPDATE x = x + 1 IN PARTITION 2 WHERE p = 2;
```

## Смотрите также

- [UPDATE](#)

## DELETE IN PARTITION

Удаляет данные в указанной партиции, соответствующие указанному выражению фильтрации.  
Реализовано как мутация [mutation](#).

Синтаксис:

```
ALTER TABLE [db.]table DELETE [IN PARTITION partition_id] WHERE filter_expr
```

## Пример

```
ALTER TABLE mt DELETE IN PARTITION 2 WHERE p = 2;
```

## Смотрите также

- [DELETE](#)

## Как задавать имя партиции в запросах ALTER

Чтобы задать нужную партицию в запросах `ALTER ... PARTITION`, можно использовать:

- Имя партиции. Посмотреть имя партиции можно в столбце `partition` системной таблицы `system.parts`. Например, `ALTER TABLE visits DETACH PARTITION 201901`
- Кортеж из выражений или констант, совпадающий (в типах) с кортежем парционарирования. В случае ключа парционарирования из одного элемента, выражение следует обернуть в функцию `tuple(...)`. Например, `ALTER TABLE visits DETACH PARTITION tuple(toYYYYMM(toDate('2019-01-25')))`.
- Строковый идентификатор партиции. Идентификатор партиции используется для именования кусков партиции на файловой системе и в ZooKeeper. В запросах `ALTER` идентификатор партиции нужно указывать в секции `PARTITION ID`, в одинарных кавычках. Например, `ALTER TABLE visits DETACH PARTITION ID '201901'`.
- Для запросов `ATTACH PART` и `DROP DETACHED PART`: чтобы задать имя куска партиции, используйте строковой литерал со значением из столбца `name` системной таблицы `system.detached_parts`. Например, `ALTER TABLE visits ATTACH PART '201901_1_1_0'`.

Использование кавычек в имени партиций зависит от типа данных столбца, по которому задано парционарирование. Например, для столбца с типом `String` имя партиции необходимо указывать в кавычках (одинарных). Для типов `Date` и `Int*` кавычки указывать не нужно.

Замечание: для таблиц старого стиля партицию можно указывать и как число 201901, и как строку '201901'. Синтаксис для таблиц нового типа более строг к типам (аналогично парсеру входного формата VALUES).

Правила, сформулированные выше, актуальны также для запросов **OPTIMIZE**. Чтобы указать единственную партицию непартиционированной таблицы, укажите **PARTITION tuple()**. Например:

```
OPTIMIZE TABLE table_not_partitioned PARTITION tuple() FINAL;
```

**IN PARTITION** указывает на партицию, для которой применяются выражения **UPDATE** или **DELETE** в результате запроса **ALTER TABLE**. Новые куски создаются только в указанной партиции. Таким образом, **IN PARTITION** помогает снизить нагрузку, когда таблица разбита на множество партиций, а вам нужно обновить данные лишь точечно.

Примеры запросов **ALTER ... PARTITION** можно посмотреть в тестах: `00502_custom_partitioning_local` и `00502_custom_partitioning_replicated_zookeeper`.

## Изменение настроек таблицы

Существуют запросы, которые изменяют настройки таблицы или сбрасывают их в значения по умолчанию. В одном запросе можно изменить сразу несколько настроек. Если настройка с указанным именем не существует, то генерируется исключение.

### Синтаксис

```
ALTER TABLE [db].name [ON CLUSTER cluster] MODIFY|RESET SETTING ...
```

### Примечание

Эти запросы могут применяться только к таблицам на движке **MergeTree**.

## MODIFY SETTING

Изменяет настройки таблицы.

### Синтаксис

```
MODIFY SETTING setting_name=value [, ...]
```

### Пример

```
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

Сбрасывает настройки таблицы в значения по умолчанию. Если настройка уже находится в состоянии по умолчанию, то никакие действия не выполняются.

### Синтаксис

```
RESET SETTING setting_name [, ...]
```

## Пример

```
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;
```

## Смотрите также

- [Настройки MergeTree таблиц](#)

# ALTER TABLE ... DELETE

```
ALTER TABLE [db.]table [ON CLUSTER cluster] DELETE WHERE filter_expr
```

Удаляет данные, соответствующие указанному выражению фильтрации. Реализовано как **мутация**.

## Note

Предфикс `ALTER TABLE` делает этот синтаксис отличным от большинства других систем, поддерживающих SQL. Он предназначен для обозначения того, что в отличие от аналогичных запросов в базах данных OLTP это тяжелая операция, не предназначенная для частого использования.

Выражение `filter_expr` должно иметь тип `UInt8`. Запрос удаляет строки в таблице, для которых это выражение принимает ненулевое значение.

Один запрос может содержать несколько команд, разделенных запятыми.

Синхронность обработки запроса определяется параметром `mutations_sync`. По умолчанию он является асинхронным.

## Смотрите также

- [Мутации](#)
- [Синхронность запросов ALTER](#)
- [mutations\\_sync setting](#)

# ALTER TABLE ... UPDATE

```
ALTER TABLE [db.]table UPDATE column1 = expr1 [, ...] WHERE filter_expr
```

Манипулирует данными, соответствующими заданному выражению фильтрации. Реализовано как **мутация**.

## Note

Префикс `ALTER TABLE` делает этот синтаксис отличным от большинства других систем, поддерживающих SQL. Он предназначен для обозначения того, что в отличие от аналогичных запросов в базах данных OLTP это тяжелая операция, не предназначенная для частого использования.

Выражение `filter_expr` должно иметь тип `UInt8`. Запрос изменяет значение указанных столбцов на вычисленное значение соответствующих выражений в каждой строке, для которой `filter_expr` принимает ненулевое значение. Вычисленные значения преобразуются к типу столбца с помощью оператора `CAST`. Изменение столбцов, которые используются при вычислении первичного ключа или ключа партиционирования, не поддерживается.

Один запрос может содержать несколько команд, разделенных запятыми.

Синхронность обработки запроса определяется параметром `mutations_sync`. По умолчанию он является асинхронным.

#### Смотрите также

- [Мутации](#)
- [Синхронность запросов ALTER](#)
- [mutations\\_sync setting](#)

## Манипуляции с ключевыми выражениями таблиц

Поддерживается операция:

```
MODIFY ORDER BY new_expression
```

Работает только для таблиц семейства `MergeTree` (в том числе [реплицированных](#)). После выполнения запроса [ключ сортировки](#) таблицы заменяется на `new_expression` (выражение или кортеж выражений). Первичный ключ при этом остаётся прежним.

Операция затрагивает только метаданные. Чтобы сохранить свойство упорядоченности кусков данных по ключу сортировки, разрешено добавлять в ключ только новые столбцы (т.е. столбцы, добавляемые командой `ADD COLUMN` в том же запросе `ALTER`), у которых нет выражения по умолчанию.

## Manipulating Sampling-Key Expressions

Синтаксис:

```
ALTER TABLE [db].name [ON CLUSTER cluster] MODIFY SAMPLE BY new_expression
```

Команда меняет [ключ сэмплирования](#) таблицы на `new_expression` (выражение или ряд выражений).

Эта команда является упрощенной в том смысле, что она изменяет только метаданные. Первичный ключ должен содержать новый ключ сэмплирования.

## Note

Это работает только для таблиц в семействе **MergeTree** (включая **реплицируемые** таблицы).

## Манипуляции с индексами

Добавить или удалить индекс можно с помощью операций

```
ALTER TABLE [db.]name ADD INDEX name expression TYPE type GRANULARITY value [FIRST|AFTER name]
ALTER TABLE [db.]name DROP INDEX name
ALTER TABLE [db.]table MATERIALIZE INDEX name IN PARTITION partition_name
```

Поддерживается только таблицами семейства \*MergeTree.

Команда **ADD INDEX** добавляет описание индексов в метаданные, а **DROP INDEX** удаляет индекс из метаданных и стирает файлы индекса с диска, поэтому они легковесные и работают мгновенно.

Если индекс появился в метаданных, то он начнет считаться в последующих слияниях и записях в таблицу, а не сразу после выполнения операции **ALTER**.

**MATERIALIZE INDEX** - перестраивает индекс в указанной партиции. Реализовано как мутация. В случае если нужно перестроить индекс над всеми данными то писать **IN PARTITION** не нужно.

Запрос на изменение индексов реплицируется, сохраняя новые метаданные в ZooKeeper и применяя изменения на всех репликах.

## Манипуляции с ограничениями (constraints)

Про ограничения подробнее написано [тут](#).

Добавить или удалить ограничение можно с помощью запросов

```
ALTER TABLE [db].name ADD CONSTRAINT constraint_name CHECK expression;
ALTER TABLE [db].name DROP CONSTRAINT constraint_name;
```

Запросы выполняют добавление или удаление метаданных об ограничениях таблицы [db].name, поэтому выполняются мгновенно.

Если ограничение появилось для непустой таблицы, то проверка ограничения для имеющихся данных не производится.

Запрос на изменение ограничений для Replicated таблиц реплицируется, сохраняя новые метаданные в ZooKeeper и применяя изменения на всех репликах.

## Манипуляции с TTL таблицы

### MODIFY TTL

Вы можете изменить **TTL** для таблицы запросом следующего вида:

```
ALTER TABLE table-name MODIFY TTL ttl-expression
```

# REMOVE TTL

Удалить табличный TTL можно запросом следующего вида:

```
ALTER TABLE table_name REMOVE TTL
```

## Пример

Создадим таблицу с табличным TTL и заполним её данными:

```
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');
```

Выполним `OPTIMIZE` для принудительной очистки по TTL:

```
OPTIMIZE TABLE table_with_ttl FINAL;
SELECT * FROM table_with_ttl;
```

В результате видно, что вторая строка удалена.

event_time	UserID	Comment
2020-12-11 12:44:57	1	username1

Удаляем табличный TTL:

```
ALTER TABLE table_with_ttl REMOVE TTL;
```

Заново вставляем удаленную строку и снова принудительно запускаем очистку по TTL с помощью `OPTIMIZE`:

```
INSERT INTO table_with_ttl VALUES (now() - INTERVAL 4 MONTH, 2, 'username2');
OPTIMIZE TABLE table_with_ttl FINAL;
SELECT * FROM table_with_ttl;
```

TTL больше нет, поэтому данные не удаляются:

event_time	UserID	Comment
2020-12-11 12:44:57	1	username1
2020-08-11 12:44:57	2	username2

## Смотрите также

- Подробнее о [свойстве TTL](#).

- Изменить столбец с TTL.

# ALTER USER

Изменяет аккаунты пользователей ClickHouse.

Синтаксис:

```
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 Ildap 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'] [, ...]
```

Для выполнения ALTER USER необходима привилегия **ALTER USER**.

## Секция GRANTEES

Определяет пользователей или роли, которым разрешено получать **привилегии** от указанного пользователя при условии, что этому пользователю также предоставлен весь необходимый доступ с использованием **GRANT OPTION**. Параметры секции GRANTEES:

- **user** — пользователь, которому разрешено получать привилегии от указанного пользователя.
- **role** — роль, которой разрешено получать привилегии от указанного пользователя.
- **ANY** — любому пользователю или любой роли разрешено получать привилегии от указанного пользователя. Используется по умолчанию.
- **NONE** — никому не разрешено получать привилегии от указанного пользователя.

Вы можете исключить любого пользователя или роль, используя выражение **EXCEPT**. Например, `ALTER USER user1 GRANTEES ANY EXCEPT user2`. Это означает, что если `user1` имеет привилегии, предоставленные с использованием **GRANT OPTION**, он сможет предоставить их любому, кроме `user2`.

## Примеры

Установить ролями по умолчанию роли, назначенные пользователю:

```
ALTER USER user DEFAULT ROLE role1, role2;
```

Если роли не были назначены пользователю, ClickHouse выбрасывает исключение.

Установить ролями по умолчанию все роли, назначенные пользователю:

```
ALTER USER user DEFAULT ROLE ALL;
```

Если роль будет впоследствии назначена пользователю, она автоматически станет ролью по умолчанию.

Установить ролями по умолчанию все назначенные пользователю роли кроме `role1` и `role2`:

```
ALTER USER user DEFAULT ROLE ALL EXCEPT role1, role2;
```

Разрешить пользователю с аккаунтом `john` предоставить свои привилегии пользователю с аккаунтом `jack`:

```
ALTER USER john GRANTEES jack;
```

## ALTER QUOTA

Изменяет [квоту](#).

Синтаксис:

```
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 [...]}]
```

Ключи `user_name`, `ip_address`, `client_key`, `client_key, user_name` и `client_key, ip_address` соответствуют полям таблицы [system.quotas](#).

Параметры `queries`, `query_selects`, `query_inserts`, `errors`, `result_rows`, `result_bytes`, `read_rows`, `read_bytes`, `execution_time` соответствуют полям таблицы [system.quotas\\_usage](#).

В секции `ON CLUSTER` можно указать кластеры, на которых создается квота, см. [Распределенные DDL запросы](#).

### Примеры

Ограничить для текущего пользователя максимальное число запросов — не более 123 запросов за каждые 15 месяцев:

```
ALTER QUOTA IF EXISTS qa FOR INTERVAL 15 month MAX queries = 123 TO CURRENT_USER;
```

Ограничить по умолчанию максимальное время выполнения запроса — не более полсекунды за каждые 30 минут, а также максимальное число запросов — не более 321 и максимальное число ошибок — не более 10 за каждые 5 кварталов:

```
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

Изменяет роли.

Синтаксис:

```
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

Изменяет политику доступа к строкам.

Синтаксис:

```
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

Изменяет профили настроек.

Синтаксис:

```
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'] [...]
```

## Манипуляции с проекциями

Доступны следующие операции с [проекциями](#):

- `ALTER TABLE [db].name ADD PROJECTION name ( SELECT <COLUMN LIST EXPR> [GROUP BY] [ORDER BY] )` — добавляет описание проекции в метаданные.
- `ALTER TABLE [db].name DROP PROJECTION name` — удаляет описание проекции из метаданных и удаляет файлы проекции с диска.
- `ALTER TABLE [db.]table MATERIALIZE PROJECTION name IN PARTITION partition_name` — перестраивает проекцию в указанной партиции. Реализовано как [мутация](#).
- `ALTER TABLE [db.]table CLEAR PROJECTION name IN PARTITION partition_name` — удаляет файлы проекции с диска без удаления описания.

Команды `ADD`, `DROP` и `CLEAR` — легковесны, поскольку они только меняют метаданные или удаляют файлы.

Также команды реплицируются, синхронизируя описания проекций в метаданных с помощью ZooKeeper.

### Note

Манипуляции с проекциями поддерживаются только для таблиц с движком **\*MergeTree** (включая **replicated** варианты).

## Выражение ALTER TABLE ... MODIFY QUERY

Вы можете изменить запрос `SELECT`, который был задан при создании **материализованного представления**, с помощью запроса '`ALTER TABLE ... MODIFY QUERY`'. Используйте его если при создании материализованного представления не использовалась секция `TO [db.]name`. Настройка `allow_experimental_alter_materialized_view_structure` должна быть включена.

Если при создании материализованного представления использовалась конструкция `TO [db.]name`, то для изменения отсоедините представление с помощью `DETACH`, измените таблицу с помощью `ALTER TABLE`, а затем снова присоедините запрос с помощью `ATTACH`.

### Пример

```
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

## Выражение ALTER LIVE VIEW

Выражение `ALTER LIVE VIEW ... REFRESH` обновляет **Live-представление**. См. раздел [Force Live View Refresh](#).

## Запросы SYSTEM

- [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]

Перегружает все **Встроенные словари**.  
По умолчанию встроенные словари выключены.  
Всегда возвращает `Ok.`, вне зависимости от результата обновления встроенных словарей.

## RELOAD DICTIONARIES

Перегружает все словари, которые были успешно загружены до этого.  
По умолчанию включена ленивая загрузка `dictionaries_lazy_load`, поэтому словари не загружаются автоматически при старте, а только при первом обращении через `dictGet` или `SELECT` к `ENGINE=Dictionary`. После этого такие словари (`LOADED`) будут перегружаться командой `system reload dictionaries`.

Всегда возвращает `Ok.`, вне зависимости от результата обновления словарей.

## RELOAD DICTIONARY `Dictionary_name`

Полностью перегружает словарь `dictionary_name`, вне зависимости от состояния словаря (`LOADED/NOT_LOADED/FAILED`).

Всегда возвращает `Ok.`, вне зависимости от результата обновления словаря.  
Состояние словаря можно проверить запросом к `system.dictionaries`.

```
SELECT name, status FROM system.dictionaries;
```

## RELOAD MODELS

Перегружает все модели **CatBoost**, если их конфигурация была обновлена, без перезагрузки сервера.

### Синтаксис

```
SYSTEM RELOAD MODELS
```

## RELOAD MODEL

Полностью перегружает модель **CatBoost** `model_name`, если ее конфигурация была обновлена, без перезагрузки сервера.

### Синтаксис

```
SYSTEM RELOAD MODEL <model_name>
```

## RELOAD FUNCTIONS

Перезагружает все зарегистрированные **исполняемые пользовательские функции** или одну из них из файла конфигурации.

### Синтаксис

```
RELOAD FUNCTIONS  
RELOAD FUNCTION function_name
```

## DROP DNS CACHE

Сбрасывает внутренний DNS кеш ClickHouse. Иногда (для старых версий ClickHouse) необходимо использовать эту команду при изменении инфраструктуры (смене IP адреса у другого ClickHouse сервера или сервера, используемого словарями).

Для более удобного (автоматического) управления кешем см. параметры `disable_internal_dns_cache`, `dns_cache_update_period`.

## DROP MARK CACHE

Сбрасывает кеш «засечек» (mark cache). Используется при разработке ClickHouse и тестах производительности.

## DROP REPLICA

Мертвые реплики можно удалить, используя следующий синтаксис:

```
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';
```

Удаляет путь реплики из ZooKeeper-а. Это полезно, когда реплика мертва и ее метаданные не могут быть удалены из ZooKeeper с помощью `DROP TABLE`, потому что такой таблицы больше нет. `DROP REPLICA` может удалить только неактивную / устаревшую реплику и не может удалить локальную реплику, используйте для этого `DROP TABLE`. `DROP REPLICA` не удаляет таблицы и не удаляет данные или метаданные с диска.

Первая команда удаляет метаданные реплики `'replica_name'` для таблицы `database.table`.

Вторая команда удаляет метаданные реплики `'replica_name'` для всех таблиц базы данных `database`.

Третья команда удаляет метаданные реплики `'replica_name'` для всех таблиц, существующих на локальном сервере (список таблиц генерируется из локальной реплики).

Четверая команда полезна для удаления метаданных мертвой реплики когда все другие реплики таблицы уже были удалены ранее, поэтому необходимо явно указать ZooKeeper путь таблицы. ZooKeeper путь это первый аргумент для `ReplicatedMergeTree` движка при создании таблицы.

## DROP UNCOMPRESSED CACHE

Сбрасывает кеш не сжатых данных. Используется при разработке ClickHouse и тестах производительности.

Для управления кешем не сжатых данных используйте следующие настройки уровня сервера `uncompressed_cache_size` и настройки уровня запрос/пользователь/профиль `use_uncompressed_cache`

## DROP COMPILED EXPRESSION CACHE

Сбрасывает кеш скомпилированных выражений. Используется при разработке ClickHouse и тестах производительности.

Скомпилированные выражения используются когда включена настройка уровня запрос/пользователь/профиль `compile-expressions`

## FLUSH LOGS

Записывает буферы логов в системные таблицы (например `system.query_log`). Позволяет не ждать 7.5 секунд при отладке.

Если буфер логов пустой, то этот запрос просто создаст системные таблицы.

## RELOAD CONFIG

Перечитывает конфигурацию настроек ClickHouse. Используется при хранении конфигурации в zookeeper.

## SHUTDOWN

Штатно завершает работу ClickHouse (аналог `service clickhouse-server stop / kill {$pid_clickhouse-server}`)

## KILL

Аварийно завершает работу ClickHouse (аналог `kill -9 {$pid_clickhouse-server}`)

## Управление распределёнными таблицами

ClickHouse может оперировать **распределёнными** таблицами. Когда пользователь вставляет данные в эти таблицы, ClickHouse сначала формирует очередь из данных, которые должны быть отправлены на узлы кластера, а затем асинхронно отправляет подготовленные данные. Вы можете управлять очередью с помощью запросов **STOP DISTRIBUTED SENDS**, **START DISTRIBUTED SENDS** и **FLUSH DISTRIBUTED**. Также есть возможность синхронно вставлять распределенные данные с помощью настройки `insert_distributed_sync`.

### STOP DISTRIBUTED SENDS

Отключает фоновую отправку при вставке данных в распределённые таблицы.

```
SYSTEM STOP DISTRIBUTED SENDS [db.]<distributed_table_name>
```

### FLUSH DISTRIBUTED

В синхронном режиме отправляет все данные на узлы кластера. Если какие-либо узлы недоступны, ClickHouse генерирует исключение и останавливает выполнение запроса. Такой запрос можно повторять до успешного завершения, что будет означать возвращение связности с остальными узлами кластера.

```
SYSTEM FLUSH DISTRIBUTED [db.]<distributed_table_name>
```

### START DISTRIBUTED SENDS

Включает фоновую отправку при вставке данных в распределенные таблицы.

```
SYSTEM START DISTRIBUTED SENDS [db.]<distributed_table_name>
```

## Managing MergeTree Tables

ClickHouse может управлять фоновыми процессами в **MergeTree** таблицах.

### STOP MERGES

Позволяет остановить фоновые мержи для таблиц семейства MergeTree:

```
SYSTEM STOP MERGES [ON VOLUME <volume_name> | [db.]merge_tree_family_table_name]
```

### Note

`DETACH / ATTACH` таблицы восстанавливает фоновые мержи для этой таблицы (даже в случае отключения фоновых мержей для всех таблиц семейства MergeTree до `DETACH`).

## START MERGES

Включает фоновые мержи для таблиц семейства MergeTree:

```
SYSTEM START MERGES [ON VOLUME <volume_name> | [db.]merge_tree_family_table_name]
```

## STOP TTL MERGES

Позволяет остановить фоновые процессы удаления старых данных основанные на [выражениях TTL](#) для таблиц семейства MergeTree:

Возвращает `Ok.` даже если указана несуществующая таблица или таблица имеет тип отличный от MergeTree. Возвращает ошибку если указана не существующая база данных:

```
SYSTEM STOP TTL MERGES [[db.]merge_tree_family_table_name]
```

## START TTL MERGES

Запускает фоновые процессы удаления старых данных основанные на [выражениях TTL](#) для таблиц семейства MergeTree:

Возвращает `Ok.` даже если указана несуществующая таблица или таблица имеет тип отличный от MergeTree. Возвращает ошибку если указана не существующая база данных:

```
SYSTEM START TTL MERGES [[db.]merge_tree_family_table_name]
```

## STOP MOVES

Позволяет остановить фоновые процессы переноса данных основанные [табличных выражениях TTL](#) с [использованием TO VOLUME или TO DISK](#) for tables in the MergeTree family:

Возвращает `Ok.` даже если указана несуществующая таблица или таблица имеет тип отличный от MergeTree. Возвращает ошибку если указана не существующая база данных:

```
SYSTEM STOP MOVES [[db.]merge_tree_family_table_name]
```

## START MOVES

Запускает фоновые процессы переноса данных основанные [табличных выражениях TTL](#) с [использованием TO VOLUME или TO DISK](#) for tables in the MergeTree family:

Возвращает `Ok.` даже если указана несуществующая таблица или таблица имеет тип отличный от MergeTree. Возвращает ошибку если указана не существующая база данных:

```
SYSTEM START MOVES [[db.]merge_tree_family_table_name]
```

## Managing ReplicatedMergeTree Tables

ClickHouse может управлять фоновыми процессами связанными с репликацией в таблицах семейства [ReplicatedMergeTree](#).

## STOP FETCHES

Позволяет остановить фоновые процессы синхронизации новыми вставленными кусками данных с

другими репликами в кластере для таблиц семейства ReplicatedMergeTree:

Всегда возвращает Ok. вне зависимости от типа таблицы и даже если таблица или база данных не существует.

```
SYSTEM STOP FETCHES [[db.]replicated_merge_tree_family_table_name]
```

## START FETCHES

Позволяет запустить фоновые процессы синхронизации новыми вставленными кусками данных с другими репликами в кластере для таблиц семейства ReplicatedMergeTree:

Всегда возвращает Ok. вне зависимости от типа таблицы и даже если таблица или база данных не существует.

```
SYSTEM START FETCHES [[db.]replicated_merge_tree_family_table_name]
```

## STOP REPLICATED SENDS

Позволяет остановить фоновые процессы отсылки новых вставленных кусков данных другим репликам в кластере для таблиц семейства ReplicatedMergeTree:

```
SYSTEM STOP REPLICATED SENDS [[db.]replicated_merge_tree_family_table_name]
```

## START REPLICATED SENDS

Позволяет запустить фоновые процессы отсылки новых вставленных кусков данных другим репликам в кластере для таблиц семейства ReplicatedMergeTree:

```
SYSTEM START REPLICATED SENDS [[db.]replicated_merge_tree_family_table_name]
```

## STOP REPLICATION QUEUES

Останавливает фоновые процессы разбора заданий из очереди репликации которая хранится в Zookeeper для таблиц семейства ReplicatedMergeTree. Возможные типы заданий - merges, fetches, mutation, DDL запросы с ON CLUSTER:

```
SYSTEM STOP REPLICATION QUEUES [[db.]replicated_merge_tree_family_table_name]
```

## START REPLICATION QUEUES

Запускает фоновые процессы разбора заданий из очереди репликации которая хранится в Zookeeper для таблиц семейства ReplicatedMergeTree. Возможные типы заданий - merges, fetches, mutation, DDL запросы с ON CLUSTER:

```
SYSTEM START REPLICATION QUEUES [[db.]replicated_merge_tree_family_table_name]
```

## SYNC REPLICA

Ждет когда таблица семейства ReplicatedMergeTree будет синхронизирована с другими репликами в кластере, будет работать до достижения receive\_timeout, если синхронизация для таблицы отключена в настоящий момент времени:

```
SYSTEM SYNC REPLICA [db.]replicated_merge_tree_family_table_name
```

После выполнения этого запроса таблица [db.]replicated\_merge\_tree\_family\_table\_name синхронизирует команды из общего реплицированного лога в свою собственную очередь репликации. Затем запрос ждет, пока реплика не обработает все синхронизированные команды.

## RESTART REPLICA

Реинициализирует состояние сессий Zookeeper для таблицы семейства ReplicatedMergeTree.

Сравнивает текущее состояние с состоянием в Zookeeper (как с эталоном) и при необходимости добавляет задачи в очередь репликации в Zookeeper.

Инициализация очереди репликации на основе данных ZooKeeper происходит так же, как при ATTACH TABLE. Некоторое время таблица будет недоступна для любых операций.

```
SYSTEM RESTART REPLICA [db.]replicated_merge_tree_family_table_name
```

## RESTORE REPLICA

Восстанавливает реплику, если метаданные в Zookeeper потеряны, но сами данные возможно существуют.

Работает только с таблицами семейства ReplicatedMergeTree и только если таблица находится в readonly-режиме.

Запрос можно выполнить если:

- потерян корневой путь ZooKeeper /;
- потерян путь реплик /replicas;
- потерян путь конкретной реплики /replicas/replica\_name/.

К реплике прикрепляются локально найденные куски, информация о них отправляется в Zookeeper. Если присутствующие в реплике до потери метаданных данные не устарели, они не скачиваются повторно с других реплик. Поэтому восстановление реплики не означает повторную загрузку всех данных по сети.

### Предупреждение

Потерянные данные в любых состояниях перемещаются в папку detached/. Куски, активные до потери данных (находившиеся в состоянии Committed), прикрепляются.

### Синтаксис

```
SYSTEM RESTORE REPLICA [db.]replicated_merge_tree_family_table_name [ON CLUSTER cluster_name]
```

Альтернативный синтаксис:

```
SYSTEM RESTORE REPLICA [ON CLUSTER cluster_name] [db.]replicated_merge_tree_family_table_name
```

### Пример

Создание таблицы на нескольких серверах. После потери корневого пути реплика таблица будет прикреплена только для чтения, так как метаданные отсутствуют. Последний запрос необходимо выполнить на каждой реплике.

```
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;
```

Альтернативный способ:

```
SYSTEM RESTORE REPLICA test ON CLUSTER cluster;
```

## RESTART REPLICAS

Реинициализация состояния ZooKeeper-сессий для всех ReplicatedMergeTree таблиц. Сравнивает текущее состояние реплики с тем, что хранится в ZooKeeper, как с источником правды, и добавляет задачи в очередь репликации в ZooKeeper, если необходимо.

## SHOW Queries

### SHOW CREATE TABLE

```
SHOW CREATE [TEMPORARY] [TABLE|DICTIONARY|VIEW] [db.]table|view [INTO OUTFILE filename] [FORMAT format]
```

Возвращает один столбец типа `String` с именем `statement`, содержащий одно значение — запрос `CREATE`, с помощью которого был создан указанный объект.

## SHOW DATABASES

Выводит список всех баз данных.

```
SHOW DATABASES [LIKE | ILIKE | NOT LIKE '<pattern>'] [LIMIT <N>] [INTO OUTFILE filename] [FORMAT format]
```

Этот запрос идентичен запросу:

```
SELECT name FROM system.databases [WHERE name LIKE | ILIKE | NOT LIKE '<pattern>'] [LIMIT <N>] [INTO OUTFILE filename] [FORMAT format]
```

## Примеры

Получение списка баз данных, имена которых содержат последовательность символов 'de':

```
SHOW DATABASES LIKE '%de%'
```

Результат:

```
name
default |
```

Получение списка баз данных, имена которых содержат последовательность символов 'de' независимо от регистра:

```
SHOW DATABASES ILIKE '%DE%'
```

Результат:

```
name
default |
```

Получение списка баз данных, имена которых не содержат последовательность символов 'de':

```
SHOW DATABASES NOT LIKE '%de%'
```

Результат:

```
name
临时_and_external_tables |
system
test
tutorial |
```

Получение первых двух строк из списка имен баз данных:

```
SHOW DATABASES LIMIT 2
```

Результат:

```
name
临时_and_external_tables |
default |
```

## Смотрите также

- [CREATE DATABASE](#)

## SHOW PROCESSLIST

```
SHOW PROCESSLIST [INTO OUTFILE filename] [FORMAT format]
```

Выводит содержимое таблицы [system.processes](#), которая содержит список запросов, выполняющихся в данный момент времени, кроме самих запросов SHOW PROCESSLIST.

Запрос `SELECT * FROM system.processes` возвращает данные обо всех текущих запросах.

Полезный совет (выполните в консоли):

```
$ watch -n1 "clickhouse-client --query='SHOW PROCESSLIST'"
```

## SHOW TABLES

Выводит список таблиц.

```
SHOW [TEMPORARY] TABLES [{FROM | IN} <db>] [LIKE | ILIKE | NOT LIKE '<pattern>'] [LIMIT <N>] [INTO OUTFILE <filename>] [FORMAT <format>]
```

Если условие FROM не указано, запрос возвращает список таблиц из текущей базы данных.

Этот запрос идентичен запросу:

```
SELECT name FROM system.tables [WHERE name LIKE | ILIKE | NOT LIKE '<pattern>'] [LIMIT <N>] [INTO OUTFILE <filename>] [FORMAT <format>]
```

## Примеры

Получение списка таблиц, имена которых содержат последовательность символов 'user':

```
SHOW TABLES FROM system LIKE '%user%'
```

Результат:

name
user_directories
users

Получение списка таблиц, имена которых содержат последовательность символов 'user' без учета регистра:

```
SHOW TABLES FROM system ILIKE '%USER%'
```

Результат:

name
user_directories
users

Получение списка таблиц, имена которых не содержат символ 's':

```
SHOW TABLES FROM system NOT LIKE '%s%'
```

Результат:

name
metric_log
metric_log_0
metric_log_1

Получение первых двух строк из списка таблиц:

```
SHOW TABLES FROM system LIMIT 2
```

Результат:

name
aggregate_function_combinators
asynchronous_metric_log

## Смотрите также

- [Create Tables](#)
- [SHOW CREATE TABLE](#)

## SHOW DICTIONARIES

Выводит список [внешних словарей](#).

```
SHOW DICTIONARIES [FROM <db>] [LIKE '<pattern>'] [LIMIT <N>] [INTO OUTFILE <filename>] [FORMAT <format>]
```

Если секция `FROM` не указана, запрос возвращает список словарей из текущей базы данных.

Аналогичный результат можно получить следующим запросом:

```
SELECT name FROM system.dictionaries WHERE database = <db> [AND name LIKE <pattern>] [LIMIT <N>] [INTO OUTFILE <filename>] [FORMAT <format>]
```

### Example

Запрос выводит первые две строки из списка таблиц в базе данных `system`, имена которых содержат `reg`.

```
SHOW DICTIONARIES FROM db LIKE '%reg%' LIMIT 2
```

name
regions
region_names

## SHOW GRANTS

Выводит привилегии пользователя.

### Синтаксис

```
SHOW GRANTS [FOR user]
```

Если пользователь не задан, запрос возвращает привилегии текущего пользователя.

## SHOW CREATE USER

Выводит параметры, использованные при [создании пользователя](#).

`SHOW CREATE USER` не возвращает пароль пользователя.

## Синтаксис

```
SHOW CREATE USER [name1 [, name2 ...] | CURRENT_USER]
```

## SHOW CREATE ROLE

Выводит параметры, использованные при [создании роли](#).

## Синтаксис

```
SHOW CREATE ROLE name1 [, name2 ...]
```

## SHOW CREATE ROW POLICY

Выводит параметры, использованные при [создании политики доступа к строкам](#).

## Синтаксис

```
SHOW CREATE [ROW] POLICY name ON [database1.]table1 [, [database2.]table2 ...]
```

## SHOW CREATE QUOTA

Выводит параметры, использованные при [создании квоты](#).

## Синтаксис

```
SHOW CREATE QUOTA [name1 [, name2 ...] | CURRENT]
```

## SHOW CREATE SETTINGS PROFILE

Выводит параметры, использованные при [создании профиля настроек](#).

## Синтаксис

```
SHOW CREATE [SETTINGS] PROFILE name1 [, name2 ...]
```

## SHOW USERS

Выводит список [пользовательских аккаунтов](#). Для просмотра параметров пользовательских аккаунтов, см. системную таблицу [system.users](#).

## Синтаксис

```
SHOW USERS
```

## SHOW ROLES

Выводит список [ролей](#). Для просмотра параметров ролей, см. системные таблицы [system.roles](#) и [system.role-grants](#).

## Синтаксис

```
SHOW [CURRENT|ENABLED] ROLES
```

## SHOW PROFILES

Выводит список [профилей настроек](#). Для просмотра других параметров профилей настроек, см. системную таблицу [settings\\_profiles](#).

### Синтаксис

```
SHOW [SETTINGS] PROFILES
```

## SHOW POLICIES

Выводит список [политик доступа к строкам](#) для указанной таблицы. Для просмотра других параметров, см. системную таблицу [system.row\\_policies](#).

### Синтаксис

```
SHOW [ROW] POLICIES [ON [db.]table]
```

## SHOW QUOTAS

Выводит список [квот](#). Для просмотра параметров квот, см. системную таблицу [system.quotas](#).

### Синтаксис

```
SHOW QUOTAS
```

## SHOW QUOTA

Выводит потребление [квоты](#) для всех пользователей или только для текущего пользователя. Для просмотра других параметров, см. системные таблицы [system.quotas\\_usage](#) и [system.quota\\_usage](#).

### Синтаксис

```
SHOW [CURRENT] QUOTA
```

## SHOW ACCESS

Выводит список всех [пользователей](#), [ролей](#), [профилей](#) и пр., а также все [привилегии](#).

### Синтаксис

```
SHOW ACCESS
```

## SHOW CLUSTER(s)

Возвращает список кластеров. Все доступные кластеры перечислены в таблице [system.clusters](#).

### Note

По запросу `SHOW CLUSTER name` вы получите содержимое таблицы `system.clusters` для этого кластера.

## Синтаксис

```
SHOW CLUSTER '<name>'  
SHOW CLUSTERS [LIKE|NOT LIKE '<pattern>'] [LIMIT <N>]
```

## Примеры

Запрос:

```
SHOW CLUSTERS;
```

Результат:

```
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 |
```

Запрос:

```
SHOW CLUSTERS LIKE 'test%' LIMIT 1;
```

Результат:

```
cluster  
test_cluster_two_shards |
```

Запрос:

```
SHOW CLUSTER 'test_shard_localhost' FORMAT Vertical;
```

Результат:

```
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

Возвращает список системных настроек и их значений. Использует данные из таблицы [system.settings](#).

## Синтаксис

```
SHOW [CHANGED] SETTINGS LIKE|ILIKE <name>
```

## Секции

При использовании `LIKE|ILIKE` можно задавать шаблон для имени настройки. Этот шаблон может содержать символы подстановки, такие как `%` или `_`. При использовании `LIKE` шаблон чувствителен к регистру, а при использовании `ILIKE` — не чувствителен.

Если используется `CHANGED`, запрос вернет только те настройки, значения которых были изменены, т.е. отличны от значений по умолчанию.

## Примеры

Запрос с использованием `LIKE`:

```
SHOW SETTINGS LIKE 'send_timeout';
```

Результат:

name	type	value
send_timeout	Seconds	300

Запрос с использованием `ILIKE`:

```
SHOW SETTINGS ILIKE '%CONNECT_timeout%'
```

Результат:

name	type	value
connect_timeout	Seconds	10
connect_timeout_with_failover_ms	Milliseconds	50
connect_timeout_with_failover_secure_ms	Milliseconds	100

Запрос с использованием `CHANGED`:

```
SHOW CHANGED SETTINGS ILIKE '%MEMORY%'
```

Результат:

name	type	value
max_memory_usage	UInt64	100000000000

## См. также

- Таблица [system.settings](#)

# GRANT

- Присваивает **привилегии** пользователям или ролям ClickHouse.
- Назначает роли пользователям или другим ролям.

Отозвать привилегию можно с помощью выражения **REVOKE**. Чтобы вывести список присвоенных привилегий, воспользуйтесь выражением **SHOW GRANTS**.

## Синтаксис присвоения привилегий

```
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** — Тип привилегии
- **role** — Роль пользователя ClickHouse.
- **user** — Пользователь ClickHouse.

**WITH GRANT OPTION** разрешает пользователю или роли выполнять запрос **GRANT**. Пользователь может выдавать только те привилегии, которые есть у него, той же или меньшей области действий.

**WITH REPLACE OPTION** заменяет все старые привилегии новыми привилегиями для **user** или **role**. Если не указано, добавьте новые привилегии для старых.

## Синтаксис назначения ролей

```
GRANT [ON CLUSTER cluster_name] role [...] TO {user | another_role | CURRENT_USER} [...] [WITH ADMIN OPTION]  
[WITH REPLACE OPTION]
```

- **role** — Роль пользователя ClickHouse.
- **user** — Пользователь ClickHouse.

**WITH ADMIN OPTION** присваивает привилегию **ADMIN OPTION** пользователю или роли.

**WITH REPLACE OPTION** заменяет все старые роли новыми ролями для пользователя **user** или **role**. Если не указано, добавьте новые роли в старые.

## Использование

Для использования **GRANT** пользователь должен иметь привилегию **GRANT OPTION**. Пользователь может выдавать привилегии только внутри области действий назначенных ему самому привилегий.

Например, администратор выдал привилегию пользователю **john**:

```
GRANT SELECT(x,y) ON db.table TO john WITH GRANT OPTION
```

Это означает, что пользователю **john** разрешено выполнять:

- **SELECT x,y FROM db.table.**
- **SELECT x FROM db.table.**
- **SELECT y FROM db.table.**

`john` не может выполнить `SELECT z FROM db.table` или `SELECT * FROM db.table`. После обработки данных запросов ClickHouse ничего не вернет — даже `x` или `y`. Единственное исключение — если таблица содержит только столбцы `x` и `y`. В таком случае ClickHouse вернет все данные.

Также у `john` есть привилегия `GRANT OPTION`. `john` может выдать другим пользователям привилегии той же или меньшей области действий из тех, которые есть у него.

При присвоении привилегий допускается использовать астериск (\*) вместо имени таблицы или базы данных. Например, запрос `GRANT SELECT ON db.* TO john` позволит пользователю `john` выполнять `SELECT` над всеми таблицами в базе данных `db`. Также вы можете опускать имя базы данных. В таком случае привилегии позволяют совершать операции над текущей базой данных. Например, запрос `GRANT SELECT ON * TO john` выдаст привилегию на выполнение `SELECT` над всеми таблицами в текущей базе данных; `GRANT SELECT ON mytable TO john` — только над таблицей `mytable` в текущей базе данных.

Доступ к базе данных `system` разрешен всегда (данная база данных используется при обработке запросов).

Вы можете присвоить несколько привилегий нескольким пользователям в одном запросе. Запрос `GRANT SELECT, INSERT ON *.* TO john, robin` позволит пользователям `john` и `robin` выполнять `INSERT` и `SELECT` над всеми таблицами всех баз данных на сервере.

## Привилегии

Привилегия — это разрешение на выполнение определенного типа запросов.

Привилегии имеют иерархическую структуру. Набор разрешенных запросов зависит от области действия привилегии.

Иерархия привилегий:

- `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
  - MYSQL
  - ODBC
  - JDBC
  - HDFS
  - S3
- dictGet

Примеры того, как трактуется данная иерархия:

- Привилегия ALTER включает все остальные ALTER\* привилегии.
- ALTER CONSTRAINT включает ALTER ADD CONSTRAINT и ALTER DROP CONSTRAINT.

Привилегии применяются на разных уровнях. Уровень определяет синтаксис присваивания привилегии.

Уровни (от низшего к высшему):

- COLUMN — Привилегия присваивается для столбца, таблицы, базы данных или глобально.
- TABLE — Привилегия присваивается для таблицы, базы данных или глобально.
- VIEW — Привилегия присваивается для представления, базы данных или глобально.
- DICTIONARY — Привилегия присваивается для словаря, базы данных или глобально.
- DATABASE — Привилегия присваивается для базы данных или глобально.
- GLOBAL — Привилегия присваивается только глобально.
- GROUP — Группирует привилегии разных уровней. При присвоении привилегии уровня GROUP присваиваются только привилегии из группы в соответствии с используемым синтаксисом.

Примеры допустимого синтаксиса:

- GRANT SELECT(x) ON db.table TO user
- GRANT SELECT ON db.\* TO user

Примеры недопустимого синтаксиса:

- GRANT CREATE USER(x) ON db.table TO user
- GRANT CREATE USER ON db.\* TO user

Специальная привилегия ALL присваивает все привилегии пользователю или роли.

По умолчанию пользователь или роль не имеют привилегий.

Отсутствие привилегий у пользователя или роли отображается как привилегия NONE.

Выполнение некоторых запросов требует определенного набора привилегий. Например, чтобы выполнить запрос **RENAME**, нужны следующие привилегии: **SELECT**, **CREATE TABLE**, **INSERT** и **DROP TABLE**.

## SELECT

Разрешает выполнять запросы **SELECT**.

Уровень: COLUMN.

### Описание

Пользователь с данной привилегией может выполнять запросы **SELECT** над определенными столбцами из определенной таблицы и базы данных. При включении в запрос других столбцов запрос ничего не вернет.

Рассмотрим следующую привилегию:

```
GRANT SELECT(x,y) ON db.table TO john
```

Данная привилегия позволяет пользователю **john** выполнять выборку данных из столбцов **x** и/или **y** в **db.table**, например, **SELECT x FROM db.table**. **john** не может выполнить **SELECT z FROM db.table** или **SELECT \* FROM db.table**. После обработки данных запросов ClickHouse ничего не вернет — даже **x** или **y**. Единственное исключение — если таблица содержит только столбцы **x** и **y**. В таком случае ClickHouse вернет все данные.

## INSERT

Разрешает выполнять запросы **INSERT**.

Уровень: COLUMN.

### Описание

Пользователь с данной привилегией может выполнять запросы **INSERT** над определенными столбцами из определенной таблицы и базы данных. При включении в запрос других столбцов запрос не добавит никаких данных.

### Пример

```
GRANT INSERT(x,y) ON db.table TO john
```

Присвоенная привилегия позволит пользователю **john** вставить данные в столбцы **x** и/или **y** в **db.table**.

## ALTER

Разрешает выполнять запросы **ALTER** в соответствии со следующей иерархией привилегий:

- **ALTER**. Уровень: COLUMN.

- ALTER TABLE. Уровень: GROUP
  - ALTER UPDATE. Уровень: COLUMN. Алиасы: UPDATE
  - ALTER DELETE. Уровень: COLUMN. Алиасы: DELETE
  - ALTER COLUMN. Уровень: GROUP
    - ALTER ADD COLUMN. Уровень: COLUMN. Алиасы: ADD COLUMN
    - ALTER DROP COLUMN. Уровень: COLUMN. Алиасы: DROP COLUMN
    - ALTER MODIFY COLUMN. Уровень: COLUMN. Алиасы: MODIFY COLUMN
    - ALTER COMMENT COLUMN. Уровень: COLUMN. Алиасы: COMMENT COLUMN
    - ALTER CLEAR COLUMN. Уровень: COLUMN. Алиасы: CLEAR COLUMN
    - ALTER RENAME COLUMN. Уровень: COLUMN. Алиасы: RENAME COLUMN
  - ALTER INDEX. Уровень: GROUP. Алиасы: INDEX
    - ALTER ORDER BY. Уровень: TABLE. Алиасы: ALTER MODIFY ORDER BY, MODIFY ORDER BY
    - ALTER SAMPLE BY. Уровень: TABLE. Алиасы: ALTER MODIFY SAMPLE BY, MODIFY SAMPLE BY
    - ALTER ADD INDEX. Уровень: TABLE. Алиасы: ADD INDEX
    - ALTER DROP INDEX. Уровень: TABLE. Алиасы: DROP INDEX
    - ALTER MATERIALIZE INDEX. Уровень: TABLE. Алиасы: MATERIALIZE INDEX
    - ALTER CLEAR INDEX. Уровень: TABLE. Алиасы: CLEAR INDEX
  - ALTER CONSTRAINT. Уровень: GROUP. Алиасы: CONSTRAINT
    - ALTER ADD CONSTRAINT. Уровень: TABLE. Алиасы: ADD CONSTRAINT
    - ALTER DROP CONSTRAINT. Уровень: TABLE. Алиасы: DROP CONSTRAINT
  - ALTER TTL. Уровень: TABLE. Алиасы: ALTER MODIFY TTL, MODIFY TTL
    - ALTER MATERIALIZE TTL. Уровень: TABLE. Алиасы: MATERIALIZE TTL
  - ALTER SETTINGS. Уровень: TABLE. Алиасы: ALTER SETTING, ALTER MODIFY SETTING, MODIFY SETTING
  - ALTER MOVE PARTITION. Уровень: TABLE. Алиасы: ALTER MOVE PART, MOVE PARTITION, MOVE PART
  - ALTER FETCH PARTITION. Уровень: TABLE. Алиасы: ALTER FETCH PART, FETCH PARTITION, FETCH PART
  - ALTER FREEZE PARTITION. Уровень: TABLE. Алиасы: FREEZE PARTITION
- ALTER VIEW Уровень: GROUP
  - ALTER VIEW REFRESH. Уровень: VIEW. Алиасы: ALTER LIVE VIEW REFRESH, REFRESH VIEW
  - ALTER VIEW MODIFY QUERY. Уровень: VIEW. Алиасы: ALTER TABLE MODIFY QUERY

Примеры того, как трактуется данная иерархия:

- Привилегия ALTER включает все остальные ALTER\* привилегии.
- ALTER CONSTRAINT включает ALTER ADD CONSTRAINT и ALTER DROP CONSTRAINT.

## **Дополнительно**

- Привилегия MODIFY SETTING позволяет изменять настройки движков таблиц. Не влияет на настройки или конфигурационные параметры сервера.

- Операция `ATTACH` требует наличие привилегии `CREATE`.
- Операция `DETACH` требует наличие привилегии `DROP`.
- Для остановки мутации с помощью `KILL MUTATION`, необходима привилегия на выполнение данной мутации. Например, чтобы остановить запрос `ALTER UPDATE`, необходима одна из привилегий: `ALTER UPDATE`, `ALTER TABLE` или `ALTER`.

## CREATE

Разрешает выполнять DDL-запросы `CREATE` и `ATTACH` в соответствии со следующей иерархией привилегий:

- `CREATE`. Уровень: GROUP
  - `CREATE DATABASE`. Уровень: DATABASE
  - `CREATE TABLE`. Уровень: TABLE
    - `CREATE TEMPORARY TABLE`. Уровень: GLOBAL
  - `CREATE VIEW`. Уровень: VIEW
  - `CREATE DICTIONARY`. Уровень: DICTIONARY

## Дополнительно

- Для удаления созданной таблицы пользователю необходима привилегия `DROP`.

## DROP

Разрешает выполнять запросы `DROP` и `DETACH` в соответствии со следующей иерархией привилегий:

- `DROP`. Уровень: GROUP
  - `DROP DATABASE`. Уровень: DATABASE
  - `DROP TABLE`. Уровень: TABLE
  - `DROP VIEW`. Уровень: VIEW
  - `DROP DICTIONARY`. Уровень: DICTIONARY

## TRUNCATE

Разрешает выполнять запросы `TRUNCATE`.

Уровень: TABLE.

## OPTIMIZE

Разрешает выполнять запросы `OPTIMIZE TABLE`.

Уровень: TABLE.

## SHOW

Разрешает выполнять запросы `SHOW`, `DESCRIBE`, `USE` и `EXISTS` в соответствии со следующей иерархией привилегий:

- SHOW. Уровень: GROUP
  - SHOW DATABASES. Уровень: DATABASE. Разрешает выполнять запросы SHOW DATABASES, SHOW CREATE DATABASE, USE <database>.
  - SHOW TABLES. Уровень: TABLE. Разрешает выполнять запросы SHOW TABLES, EXISTS <table>, CHECK <table>.
  - SHOW COLUMNS. Уровень: COLUMN. Разрешает выполнять запросы SHOW CREATE TABLE, DESCRIBE.
  - SHOW DICTIONARIES. Уровень: DICTIONARY. Разрешает выполнять запросы SHOW DICTIONARIES, SHOW CREATE DICTIONARY, EXISTS <dictionary>.

## **Дополнительно**

У пользователя есть привилегия SHOW, если ему присвоена любая другая привилегия по отношению к определенной таблице, словарю или базе данных.

## **KILL QUERY**

Разрешает выполнять запросы KILL в соответствии со следующей иерархией привилегий:

Уровень: GLOBAL.

## **Дополнительно**

KILL QUERY позволяет пользователю останавливать запросы других пользователей.

## **ACCESS MANAGEMENT**

Разрешает пользователю выполнять запросы на управление пользователями, ролями и политиками доступа к строкам.

- **ACCESS MANAGEMENT**. Уровень: GROUP
  - **CREATE USER**. Уровень: GLOBAL
  - **ALTER USER**. Уровень: GLOBAL
  - **DROP USER**. Уровень: GLOBAL
  - **CREATE ROLE**. Уровень: GLOBAL
  - **ALTER ROLE**. Уровень: GLOBAL
  - **DROP ROLE**. Уровень: GLOBAL
  - **ROLE ADMIN**. Уровень: GLOBAL
  - **CREATE ROW POLICY**. Уровень: GLOBAL. Алиасы: CREATE POLICY
  - **ALTER ROW POLICY**. Уровень: GLOBAL. Алиасы: ALTER POLICY
  - **DROP ROW POLICY**. Уровень: GLOBAL. Алиасы: DROP POLICY
  - **CREATE QUOTA**. Уровень: GLOBAL
  - **ALTER QUOTA**. Уровень: GLOBAL
  - **DROP QUOTA**. Уровень: GLOBAL
  - **CREATE SETTINGS PROFILE**. Уровень: GLOBAL. Алиасы: CREATE PROFILE
  - **ALTER SETTINGS PROFILE**. Уровень: GLOBAL. Алиасы: ALTER PROFILE
  - **DROP SETTINGS PROFILE**. Уровень: GLOBAL. Алиасы: DROP PROFILE
  - **SHOW ACCESS**. Уровень: GROUP
    - **SHOW\_USERS**. Уровень: GLOBAL. Алиасы: SHOW CREATE USER
    - **SHOW\_ROLES**. Уровень: GLOBAL. Алиасы: SHOW CREATE ROLE
    - **SHOW\_ROW\_POLICIES**. Уровень: GLOBAL. Алиасы: SHOW POLICIES, SHOW CREATE ROW POLICY, SHOW CREATE POLICY
    - **SHOW\_QUOTAS**. Уровень: GLOBAL. Алиасы: SHOW CREATE QUOTA
    - **SHOW\_SETTINGS\_PROFILES**. Уровень: GLOBAL. Алиасы: SHOW PROFILES, SHOW CREATE SETTINGS PROFILE, SHOW CREATE PROFILE

Привилегия **ROLE ADMIN** разрешает пользователю назначать и отзывать любые роли, включая те, которые не назначены пользователю с опцией администратора.

## SYSTEM

Разрешает выполнять запросы **SYSTEM** в соответствии со следующей иерархией привилегий:

- SYSTEM. Уровень: GROUP
  - SYSTEM SHUTDOWN. Уровень: GLOBAL. Алиасы: SYSTEM KILL, SHUTDOWN
  - SYSTEM DROP CACHE. Алиасы: DROP CACHE
    - SYSTEM DROP DNS CACHE. Уровень: GLOBAL. Алиасы: SYSTEM DROP DNS, DROP DNS CACHE, DROP DNS
    - SYSTEM DROP MARK CACHE. Уровень: GLOBAL. Алиасы: SYSTEM DROP MARK, DROP MARK CACHE, DROP MARKS
  - SYSTEM DROP UNCOMPRESSED CACHE. Уровень: GLOBAL. Алиасы: SYSTEM DROP UNCOMPRESSED, DROP UNCOMPRESSED CACHE, DROP UNCOMPRESSED
- SYSTEM RELOAD. Уровень: GROUP
  - SYSTEM RELOAD CONFIG. Уровень: GLOBAL. Алиасы: RELOAD CONFIG
  - SYSTEM RELOAD DICTIONARY. Уровень: GLOBAL. Алиасы: SYSTEM RELOAD DICTIONARIES, RELOAD DICTIONARY, RELOAD DICTIONARIES
    - SYSTEM RELOAD EMBEDDED DICTIONARIES. Уровень: GLOBAL. Алиасы: RELOAD EMBEDDED DICTIONARIES
- SYSTEM MERGES. Уровень: TABLE. Алиасы: SYSTEM STOP MERGES, SYSTEM START MERGES, STOP MERGES, START MERGES
- SYSTEM TTL MERGES. Уровень: TABLE. Алиасы: SYSTEM STOP TTL MERGES, SYSTEM START TTL MERGES, STOP TTL MERGES, START TTL MERGES
- SYSTEM FETCHES. Уровень: TABLE. Алиасы: SYSTEM STOP FETCHES, SYSTEM START FETCHES, STOP FETCHES, START FETCHES
- SYSTEM MOVES. Уровень: TABLE. Алиасы: SYSTEM STOP MOVES, SYSTEM START MOVES, STOP MOVES, START MOVES
- SYSTEM SENDS. Уровень: GROUP. Алиасы: SYSTEM STOP SENDS, SYSTEM START SENDS, STOP SENDS, START SENDS
  - SYSTEM DISTRIBUTED SENDS. Уровень: TABLE. Алиасы: SYSTEM STOP DISTRIBUTED SENDS, SYSTEM START DISTRIBUTED SENDS, STOP DISTRIBUTED SENDS, START DISTRIBUTED SENDS
  - SYSTEM REPLICATED SENDS. Уровень: TABLE. Алиасы: SYSTEM STOP REPLICATED SENDS, SYSTEM START REPLICATED SENDS, STOP REPLICATED SENDS, START REPLICATED SENDS
- SYSTEM REPLICATION QUEUES. Уровень: TABLE. Алиасы: SYSTEM STOP REPLICATION QUEUES, SYSTEM START REPLICATION QUEUES, STOP REPLICATION QUEUES, START REPLICATION QUEUES
- SYSTEM SYNC REPLICA. Уровень: TABLE. Алиасы: SYNC REPLICA
- SYSTEM RESTART REPLICA. Уровень: TABLE. Алиасы: RESTART REPLICA
- SYSTEM FLUSH. Уровень: GROUP
  - SYSTEM FLUSH DISTRIBUTED. Уровень: TABLE. Алиасы: FLUSH DISTRIBUTED
  - SYSTEM FLUSH LOGS. Уровень: GLOBAL. Алиасы: FLUSH LOGS

Привилегия SYSTEM RELOAD EMBEDDED DICTIONARIES имплицитно присваивается привилегией SYSTEM RELOAD DICTIONARY ON \*.\*.

## INTROSPECTION

Разрешает использовать функции [интроспекции](#).

- **INTROSPECTION**. Уровень: GROUP. Алиасы: INTROSPECTION FUNCTIONS
  - `addressToLine`. Уровень: GLOBAL
  - `addressToSymbol`. Уровень: GLOBAL
  - `demangle`. Уровень: GLOBAL

## SOURCES

Разрешает использовать внешние источники данных. Применяется к движкам таблиц и табличным функциям.

- **SOURCES**. Уровень: GROUP
  - `FILE`. Уровень: GLOBAL
  - `URL`. Уровень: GLOBAL
  - `REMOTE`. Уровень: GLOBAL
  - `YSQL`. Уровень: GLOBAL
  - `ODBC`. Уровень: GLOBAL
  - `JDBC`. Уровень: GLOBAL
  - `HDFS`. Уровень: GLOBAL
  - `S3`. Уровень: GLOBAL

Привилегия `SOURCES` разрешает использование всех источников. Также вы можете присвоить привилегию для каждого источника отдельно. Для использования источников необходимы дополнительные привилегии.

Примеры:

- Чтобы создать таблицу с движком MySQL, необходимы привилегии `CREATE TABLE (ON db.table_name)` и `MySQL`.
- Чтобы использовать табличную функцию `mysql`, необходимы привилегии `CREATE TEMPORARY TABLE` и `MySQL`.

## dictGet

- `dictGet`. Алиасы: `dictHas`, `dictGetHierarchy`, `dictIsIn`

Разрешает вызывать функции `dictGet`, `dictHas`, `dictGetHierarchy`, `dictIsIn`.

Уровень: DICTIONARY.

### Примеры

- `GRANT dictGet ON mydb.mydictionary TO john`
- `GRANT dictGet ON mydictionary TO john`

## ALL

Присваивает пользователю или роли все привилегии на объект с регулируемым доступом.

## NONE

Не присваивает никаких привилегий.

## ADMIN OPTION

Привилегия **ADMIN OPTION** разрешает пользователю назначать свои роли другому пользователю.

## EXPLAIN

Выводит план выполнения запроса.

Синтаксис:

```
EXPLAIN [AST | SYNTAX | PLAN | PIPELINE] [setting = value, ...] SELECT ... [FORMAT ...]
```

Пример:

```
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)
```

## Типы EXPLAIN

- **AST** — абстрактное синтаксическое дерево.
- **SYNTAX** — текст запроса после оптимизации на уровне AST.
- **PLAN** — план выполнения запроса.
- **PIPELINE** — конвейер выполнения запроса.

## EXPLAIN AST

Дамп AST запроса. Поддерживает все типы запросов, не только **SELECT**.

Примеры:

```
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

Возвращает текст запроса после применения синтаксических оптимизаций.

Пример:

```
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

Дамп шагов выполнения запроса.

Настройки:

- `header` — выводит выходной заголовок для шага. По умолчанию: 0.
- `description` — выводит описание шага. По умолчанию: 1.
- `indexes` — показывает используемые индексы, количество отфильтрованных кусков и гранул для каждого примененного индекса. По умолчанию: 0. Поддерживается для таблиц семейства **MergeTree**.
- `actions` — выводит подробную информацию о действиях, выполняемых на данном шаге. По умолчанию: 0.
- `json` — выводит шаги выполнения запроса в виде строки в формате **JSON**. По умолчанию: 0. Чтобы избежать ненужного экранирования, рекомендуется использовать формат **TSVRaw**.

Пример:

```
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)
```

## Примечание

Оценка стоимости выполнения шага и запроса не поддерживается.

При `json = 1` шаги выполнения запроса выводятся в формате JSON. Каждый узел — это словарь, в котором всегда есть ключи `Node Type` и `Plans`. `Node Type` — это строка с именем шага. `Plans` — это массив с описаниями дочерних шагов. Другие дополнительные ключи могут быть добавлены в зависимости от типа узла и настроек.

Пример:

```
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"
                }
              ]
            }
          ]
        }
      ]
    }
]
```

При `description = 1` к шагу добавляется ключ `Description`:

```
{
  "Node Type": "ReadFromStorage",
  "Description": "SystemOne"
}
```

При `header = 1` к шагу добавляется ключ `Header` в виде массива столбцов.

Пример:

```
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"  
                }  
              ]  
            }  
          ]  
        }  
      ]  
    }  
  ]
```

При `indexes = 1` добавляется ключ `Indexes`. Он содержит массив используемых индексов. Каждый индекс описывается как строка в формате JSON с ключом `Type` (`MinMax`, `Partition`, `PrimaryKey` или `Skip`) и дополнительные ключи:

- `Name` — имя индекса (на данный момент используется только для индекса `Skip`).
- `Keys` — массив столбцов, используемых индексом.
- `Condition` — строка с используемым условием.
- `Description` — индекс (на данный момент используется только для индекса `Skip`).
- `Initial Parts` — количество кусков до применения индекса.
- `Selected Parts` — количество кусков после применения индекса.
- `Initial Granules` — количество гранул до применения индекса.
- `Selected Granules` — количество гранул после применения индекса.

Пример:

```

"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
}
]

```

При `actions = 1` добавляются ключи, зависящие от типа шага.

Пример:

```
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

Настройки:

- `header` — выводит заголовок для каждого выходного порта. По умолчанию: 0.
- `graph` — выводит граф, описанный на языке **DOT**. По умолчанию: 0.
- `compact` — выводит график в компактном режиме, если включена настройка `graph`. По умолчанию: 1.

Пример:

```
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

Отображает оценки числа строк, засечек и кусков, которые будут прочитаны при выполнении запроса. Применяется для таблиц семейства **MergeTree**.

### Пример

Создадим таблицу:

```
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;
```

Запрос:

```
EXPLAIN ESTIMATE SELECT * FROM ttt;
```

Результат:

database	table	parts	rows	marks
default	ttt	1	128	8

## REVOKE

Отзывает привилегии у пользователей или ролей.

### Синтаксис

#### Отзыв привилегий у пользователей

```
REVOKE [ON CLUSTER cluster_name] privilege[(column_name [...])] [...] ON {db.table|db.*|*.*|table|*} FROM {user | CURRENT_USER} [...] | ALL | ALL EXCEPT {user | CURRENT_USER} [...]
```

#### Отзыв ролей у пользователей

```
REVOKE [ON CLUSTER cluster_name] [ADMIN OPTION FOR] role [...] FROM {user | role | CURRENT_USER} [...] | ALL | ALL EXCEPT {user_name | role_name | CURRENT_USER} [...]
```

## Описание

Для отзыва привилегий можно использовать привилегию более широкой области действия.

Например, если у пользователя есть привилегия `SELECT (x,y)`, администратор может отозвать ее с помощью одного из запросов: `REVOKE SELECT(x,y) ...`, `REVOKE SELECT * ...` или даже `REVOKE ALL PRIVILEGES ...`

### Частичный отзыв

Вы можете отзывать часть привилегии. Например, если у пользователя есть привилегия `SELECT *.*`, вы можете отзывать привилегию на чтение данных из какой-то таблицы или базы данных.

### Примеры

Присвоить пользователю `john` привилегию на `SELECT` из всех баз данных кроме `accounts`:

```
GRANT SELECT ON *.* TO john;
REVOKE SELECT ON accounts.* FROM john;
```

Присвоить пользователю `mira` привилегию на `SELECT` из всех столбцов таблицы `accounts.staff` кроме столбца `wage`:

```
GRANT SELECT ON accounts.staff TO mira;
REVOKE SELECT(wage) ON accounts.staff FROM mira;
```

## ATTACH

Выполняет подключение таблицы или словаря, например, при перемещении базы данных на другой сервер.

### Синтаксис

```
ATTACH TABLE|DICTIONARY [IF NOT EXISTS] [db.]name [ON CLUSTER cluster] ...
```

Запрос не создаёт данные на диске, а предполагает, что данные уже лежат в соответствующих местах, и всего лишь добавляет информацию о таблице или словаре на сервер. После выполнения запроса `ATTACH` сервер будет знать о существовании таблицы или словаря.

Если таблица перед этим была отключена при помощи (`DETACH`), т.е. её структура известна, можно использовать сокращенную форму записи без определения структуры.

## Присоединение существующей таблицы

### Синтаксис

```
ATTACH TABLE [IF NOT EXISTS] [db.]name [ON CLUSTER cluster]
```

Этот запрос используется при старте сервера. Сервер хранит метаданные таблиц в виде файлов с запросами `ATTACH`, которые он просто исполняет при запуске (за исключением некоторых системных таблиц, которые явно создаются на сервере).

Если таблица была отключена перманентно, она не будет подключена обратно во время старта сервера, так что нужно явно использовать запрос `ATTACH`, чтобы подключить ее.

## Создание новой таблицы и присоединение данных

### С указанием пути к табличным данным

Запрос создает новую таблицу с указанной структурой и присоединяет табличные данные из соответствующего каталога в `user_files`.

### Синтаксис

```
ATTACH TABLE name FROM 'path/to/data/' (col1 Type1, ...)
```

### Пример

Запрос:

```
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;
```

Результат:

s	n
test	42

## С указанием UUID таблицы

Этот запрос создает новую таблицу с указанной структурой и присоединяет данные из таблицы с указанным UUID.

Запрос поддерживается только движком баз данных [Atomic](#).

### Синтаксис

```
ATTACH TABLE name UUID '<uuid>' (col1 Type1, ...)
```

## Присоединение существующего словаря

Присоединяет ранее отключенный словарь.

### Синтаксис

```
ATTACH DICTIONARY [IF NOT EXISTS] [db.]name [ON CLUSTER cluster]
```

## CHECK TABLE Statement

Проверяет таблицу на повреждение данных.

```
CHECK TABLE [db.]name
```

Запрос `CHECK TABLE` сравнивает текущие размеры файлов (в которых хранятся данные из колонок) с ожидаемыми значениями. Если значения не совпадают, данные в таблице считаются поврежденными. Искажение возможно, например, из-за сбоя при записи данных.

Ответ содержит колонку `result`, содержащую одну строку с типом [Boolean](#). Допустимые значения:

- 0 - данные в таблице повреждены;
- 1 - данные не повреждены.

Запрос `CHECK TABLE` поддерживает следующие движки таблиц:

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

При попытке выполнить запрос с таблицами с другими табличными движками, ClickHouse генерирует исключение.

В движках \*Log не предусмотрено автоматическое восстановление данных после сбоя. Используйте запрос `CHECK TABLE`, чтобы своевременно выявлять повреждение данных.

## Проверка таблиц семейства MergeTree

Для таблиц семейства MergeTree если `check_query_single_value_result` = 0, запрос `CHECK TABLE` возвращает статус каждого куска данных таблицы на локальном сервере.

```
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	

Если `check_query_single_value_result` = 0, запрос `CHECK TABLE` возвращает статус таблицы в целом.

```
SET check_query_single_value_result = 1;
CHECK TABLE test_table;
```

result
1

## Что делать, если данные повреждены

В этом случае можно скопировать оставшиеся неповрежденные данные в другую таблицу. Для этого:

- Создайте новую таблицу с такой же структурой, как у поврежденной таблицы. Для этого выполните запрос `CREATE TABLE <new_table_name> AS <damaged_table_name>`.
- Установите значение параметра `max_threads` в 1. Это нужно для того, чтобы выполнить следующий запрос в одном потоке. Установить значение параметра можно через запрос: `SET max_threads = 1`.
- Выполните запрос `INSERT INTO <new_table_name> SELECT * FROM <damaged_table_name>`. В результате неповрежденные данные будут скопированы в другую таблицу. Обратите внимание, будут скопированы только те данные, которые следуют до поврежденного участка.
- Перезапустите `clickhouse-client`, чтобы вернуть предыдущее значение параметра `max_threads`.

## Прочие виды запросов

- [ATTACH](#)
- [CHECK TABLE](#)
- [DESCRIBE TABLE](#)
- [DETACH](#)
- [DROP](#)
- [EXISTS](#)

- **KILL**
- **OPTIMIZE**
- **RENAME**
- **SET**
- **SET ROLE**
- **TRUNCATE**
- **USE**

## DESCRIBE TABLE

Возвращает описание столбцов таблицы.

### Синтаксис

```
DESC|DESCRIBE TABLE [db.]table [INTO OUTFILE filename] [FORMAT format]
```

Запрос **DESCRIBE** для каждого столбца таблицы возвращает строку со следующими значениями типа **String**:

- **name** — имя столбца;
- **type** — тип столбца;
- **default\_type** — вид **выражения для значения по умолчанию**: **DEFAULT**, **MATERIALIZED** или **ALIAS**. Если значение по умолчанию не задано, то возвращается пустая строка;
- **default\_expression** — значение, заданное в секции **DEFAULT**;
- **comment** — **комментарий**;
- **codec\_expression** — **кодек**, который применяется к столбцу;
- **ttl\_expression** — выражение **TTL**;
- **is\_subcolumn** — флаг, который равен **1** для внутренних подстолбцов. Он появляется в результате, только если описание подстолбцов разрешено настройкой **describe\_include\_subcolumns**.

Каждый столбец **Nested** структур описывается отдельно. Перед его именем ставится имя родительского столбца с точкой.

Чтобы отобразить внутренние подстолбцы других типов данных, нужно включить настройку **describe\_include\_subcolumns**.

### Пример

Запрос:

```
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;
```

Результат:

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)				

Второй запрос дополнительно выводит информацию о подстолбцах:

name	type	default_type	default_expression	comment	codec_expre
ssion	ttl_expression	is_subcolumn			
id	UInt64	DEFAULT	'unknown'		ZSTD(1) 0
text	String				
user	Tuple(name String, age UInt8)				
user.name	String				1 0
user.age	UInt8				1 0

## См. также

- настройка [describe\\_include\\_subcolumns](#).

# DETACH

Заставляет сервер "забыть" о существовании таблицы, материализованного представления или словаря.

## Синтаксис

```
DETACH TABLE|VIEW|DICTIONARY [IF EXISTS] [db.]name [ON CLUSTER cluster] [PERMANENTLY]
```

Такой запрос не удаляет ни данные, ни метаданные таблицы, материализованного представления или словаря. Если отключение не было перманентным (запрос без ключевого слова `PERMANENTLY`), то при следующем запуске сервер прочитает метаданные и снова узнает о таблице/представлении/словаре. Если сущность была отключена перманентно, то сервер не подключит их обратно автоматически.

Независимо от того, каким способом таблица была отключена, ее можно подключить обратно с помощью запроса [ATTACH](#). Системные log таблицы также могут быть подключены обратно (к примеру, `query_log`, `text_log` и др.). Другие системные таблицы не могут быть подключены обратно, но на следующем запуске сервер снова "вспомнит" об этих таблицах.

[ATTACH MATERIALIZED VIEW](#) не может быть использован с кратким синтаксисом (без `SELECT`), но можно подключить представление с помощью запроса [ATTACH TABLE](#).

Обратите внимание, что нельзя перманентно отключить таблицу, которая уже временно отключена. Для этого ее сначала надо подключить обратно, а затем снова отключить перманентно.

Также нельзя использовать [DROP](#) с отключенной таблицей или создавать таблицу с помощью [CREATE TABLE](#) с таким же именем, как у отключенной таблицы. Еще нельзя заменить отключенную таблицу другой с помощью запроса [RENAME TABLE](#).

## Пример

Создание таблицы:

Запрос:

```
CREATE TABLE test ENGINE = Log AS SELECT * FROM numbers(10);
SELECT * FROM test;
```

Результат:

number
0
1
2
3
4
5
6
7
8
9

Отключение таблицы:

Запрос:

```
DETACH TABLE test;
SELECT * FROM test;
```

Результат:

```
Received exception from server (version 21.4.1):
Code: 60. DB::Exception: Received from localhost:9000. DB::Exception: Table default.test doesn't exist.
```

## Смотрите также

- [Материализованные представления](#)
- [Словари](#)

## DROP

Удаляет существующий объект. Если указано `IF EXISTS` - не выдавать ошибку, если объекта не существует.

### DROP DATABASE

Удаляет все таблицы в базе данных `db`, затем удаляет саму базу данных `db`.

Синтаксис:

```
DROP DATABASE [IF EXISTS] db [ON CLUSTER cluster]
```

### DROP TABLE

Удаляет таблицу.

Синтаксис:

```
DROP [TEMPORARY] TABLE [IF EXISTS] [db.]name [ON CLUSTER cluster]
```

## DROP DICTIONARY

Удаляет словарь.

Синтаксис:

```
DROP DICTIONARY [IF EXISTS] [db.]name
```

## DROP USER

Удаляет пользователя.

Синтаксис:

```
DROP USER [IF EXISTS] name [,...] [ON CLUSTER cluster_name]
```

## DROP ROLE

Удаляет роль. При удалении роль отзывается у всех объектов системы доступа, которым она присвоена.

Синтаксис:

```
DROP ROLE [IF EXISTS] name [,...] [ON CLUSTER cluster_name]
```

## DROP ROW POLICY

Удаляет политику доступа к строкам. При удалении политика отзывается у всех объектов системы доступа, которым она присвоена.

Синтаксис:

```
DROP [ROW] POLICY [IF EXISTS] name [,...] ON [database.]table [,...] [ON CLUSTER cluster_name]
```

## DROP QUOTA

Удаляет квоту. При удалении квота отзывается у всех объектов системы доступа, которым она присвоена.

Синтаксис:

```
DROP QUOTA [IF EXISTS] name [,...] [ON CLUSTER cluster_name]
```

## DROP SETTINGS PROFILE

Удаляет профиль настроек. При удалении профиль отзывается у всех объектов системы доступа, которым он присвоен.

Синтаксис:

```
DROP [SETTINGS] PROFILE [IF EXISTS] name [...] [ON CLUSTER cluster_name]
```

## DROP VIEW

Удаляет представление. Представления могут быть удалены и командой `DROP TABLE`, но команда `DROP VIEW` проверяет, что `[db.]name` является представлением.

**Синтаксис:**

```
DROP VIEW [IF EXISTS] [db.]name [ON CLUSTER cluster]
```

## DROP FUNCTION

Удаляет пользовательскую функцию, созданную с помощью [CREATE FUNCTION](#).

Удалить системные функции нельзя.

**Синтаксис**

```
DROP FUNCTION [IF EXISTS] function_name
```

## Пример

```
CREATE FUNCTION linear_equation AS (x, k, b) -> k*x + b;  
DROP FUNCTION linear_equation;
```

## EXISTS

```
EXISTS [TEMPORARY] TABLE [db.]name [INTO OUTFILE filename] [FORMAT format]
```

Возвращает один столбец типа `UInt8`, содержащий одно значение - 0, если таблицы или БД не существует и 1, если таблица в указанной БД существует.

## KILL

Существует два вида операторов `KILL`: `KILL QUERY` и `KILL MUTATION`

### KILL QUERY

```
KILL QUERY [ON CLUSTER cluster]  
WHERE <where expression to SELECT FROM system.processes query>  
[SYNC|ASYNC|TEST]  
[FORMAT format]
```

Пытается принудительно остановить исполняющиеся в данный момент запросы.

Запросы для принудительной остановки выбираются из таблицы `system.processes` с помощью условия, указанного в секции `WHERE` запроса `KILL`.

**Примеры**

```
-- Принудительно останавливает все запросы с указанным query_id:  
KILL QUERY WHERE query_id='2-857d-4a57-9ee0-327da5d60a90'
```

```
-- Синхронно останавливает все запросы пользователя 'username':  
KILL QUERY WHERE user='username' SYNC
```

Readonly-пользователи могут останавливать только свои запросы.

По умолчанию используется асинхронный вариант запроса (ASYNC), который не дожидается подтверждения остановки запросов.

Синхронный вариант (SYNC) ожидает остановки всех запросов и построчно выводит информацию о процессах по ходу их остановки.

Ответ содержит колонку `kill_status`, которая может принимать следующие значения:

1. 'finished' - запрос был успешно остановлен;
2. 'waiting' - запросу отправлен сигнал завершения, ожидается его остановка;
3. остальные значения описывают причину невозможности остановки запроса.

Тестовый вариант запроса (TEST) только проверяет права пользователя и выводит список запросов для остановки.

## KILL MUTATION

```
KILL MUTATION [ON CLUSTER cluster]  
WHERE <where expression to SELECT FROM system.mutations query>  
[TEST]  
[FORMAT format]
```

Пытается остановить выполняющиеся в данный момент **мутации**. Мутации для остановки выбираются из таблицы `system.mutations` с помощью условия, указанного в секции WHERE запроса KILL.

Тестовый вариант запроса (TEST) только проверяет права пользователя и выводит список запросов для остановки.

Примеры:

```
-- Останавливает все мутации одной таблицы:  
KILL MUTATION WHERE database = 'default' AND table = 'table'
```

```
-- Останавливает конкретную мутацию:  
KILL MUTATION WHERE database = 'default' AND table = 'table' AND mutation_id = 'mutation_3.txt'
```

Запрос полезен в случаях, когда мутация не может выполниться до конца (например, если функция в запросе мутации бросает исключение на данных таблицы).

Данные, уже изменённые мутацией, остаются в таблице (отката на старую версию данных не происходит).

## OPTIMIZE

Запрос пытается запустить внеплановое слияние кусков данных для таблиц.

### Внимание

`OPTIMIZE` не устраняет причину появления ошибки `Too many parts`.

## Синтаксис

```
OPTIMIZE TABLE [db.]name [ON CLUSTER cluster] [PARTITION partition | PARTITION ID 'partition_id'] [FINAL]
[DEDUPLICATE [BY expression]]
```

Может применяться к таблицам семейства [MergeTree](#), [MaterializedView](#) и [Buffer](#). Другие движки таблиц не поддерживаются.

Если запрос `OPTIMIZE` применяется к таблицам семейства [ReplicatedMergeTree](#), ClickHouse создаёт задачу на слияние и ожидает её исполнения на всех репликах (если значение настройки `replication_alter_partitions_sync` равно 2) или на текущей реплике (если значение настройки `replication_alter_partitions_sync` равно 1).

- По умолчанию, если запросу `OPTIMIZE` не удалось выполнить слияние, то ClickHouse не оповещает клиента. Чтобы включить оповещения, используйте настройку `optimize_throw_if_noop`.
- Если указать `PARTITION`, то оптимизация выполняется только для указанной партиции. [Как задавать имя партиции в запросах](#).
- Если указать `FINAL`, то оптимизация выполняется даже в том случае, если все данные уже лежат в одном куске данных. Кроме того, слияние является принудительным, даже если выполняются параллельные слияния.
- Если указать `DEDUPLICATE`, то произойдет схлопывание полностью одинаковых строк (сравниваются значения во всех столбцах), имеет смысл только для движка [MergeTree](#).

Вы можете указать время ожидания (в секундах) выполнения запросов `OPTIMIZE` для неактивных реплик с помощью настройки `replication_wait_for_inactive_replica_timeout`.

## Примечание

Если значение настройки `replication_alter_partitions_sync` равно 2 и некоторые реплики не активны больше времени, заданного настройкой `replication_wait_for_inactive_replica_timeout`, то генерируется исключение `UNFINISHED`.

## Выражение BY

Чтобы выполнить дедупликацию по произвольному набору столбцов, вы можете явно указать список столбцов или использовать любую комбинацию подстановки `*`, выражений `COLUMNS` и `EXCEPT`.

Список столбцов для дедупликации должен включать все столбцы, указанные в условиях сортировки (первичный ключ и ключ сортировки), а также в условияхパーティционирования (ключパーティционирования).

## Примечание

Обратите внимание, что символ подстановки `*` обрабатывается так же, как и в запросах `SELECT`: столбцы [MATERIALIZED](#) и [ALIAS](#) не включаются в результат.

Если указать пустой список или выражение, которое возвращает пустой список, то сервер вернет ошибку. Запрос вида `DEDUPLICATE BY aliased_value` также вернет ошибку.

## Синтаксис

```
OPTIMIZE TABLE table DEDUPLICATE; -- по всем столбцам
OPTIMIZE TABLE table DEDUPLICATE BY *; -- исключаются столбцы MATERIALIZED и ALIAS
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);
```

## Примеры

Рассмотрим таблицу:

```
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;
```

Результат:

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

Если в запросе не указаны столбцы, по которым нужно дедуплицировать, то учитываются все столбцы таблицы. Стока удаляется только в том случае, если все значения во всех столбцах равны соответствующим значениям в другой строке.

```
OPTIMIZE TABLE example FINAL DEDUPLICATE;
```

```
SELECT * FROM example;
```

Результат:

primary_key	secondary_key	value	partition_key
1	1	2	2
0	0	0	0
1	1	2	3
1	1	3	3

Если столбцы в запросе указаны через \*, то дедупликация пройдет по всем столбцам, кроме ALIAS и MATERIALIZED. Для таблицы example будут учтены: primary\_key, secondary\_key, value и partition\_key.

```
OPTIMIZE TABLE example FINAL DEDUPLICATE BY *;
```

```
SELECT * FROM example;
```

Результат:

primary_key	secondary_key	value	partition_key
1	1	2	2
0	0	0	0
1	1	2	3
1	1	3	3

Дедупликация по всем столбцам, кроме ALIAS и MATERIALIZED (BY \*), и с исключением столбца value: primary\_key, secondary\_key и partition\_key.

```
OPTIMIZE TABLE example FINAL DEDUPLICATE BY * EXCEPT value;
```

```
SELECT * FROM example;
```

Результат:

primary_key	secondary_key	value	partition_key
1	1	2	2
0	0	0	0

Дедупликация по столбцам primary\_key, secondary\_key и partition\_key.

```
OPTIMIZE TABLE example FINAL DEDUPLICATE BY primary_key, secondary_key, partition_key;
```

```
SELECT * FROM example;
```

Результат:

primary_key	secondary_key	value	partition_key
1	1	2	2
0	0	0	0
1	1	2	3

Дедупликация по любому столбцу, который соответствует регулярному выражению `*_key`: `primary_key`, `secondary_key` и `partition_key`.

```
OPTIMIZE TABLE example FINAL DEDUPLICATE BY COLUMNS('.*_key');
```

```
SELECT * FROM example;
```

Результат:

primary_key	secondary_key	value	partition_key
0	0	0	0
1	1	2	2
1	1	2	3

## RENAME

Переименовывает базы данных, таблицы или словари. Несколько сущностей могут быть переименованы в одном запросе.

Обратите внимание, что запрос `RENAME` с несколькими сущностями это неатомарная операция. Чтобы обменять имена атомарно, используйте выражение `EXCHANGE`.

### Примечание

Запрос `RENAME` поддерживается только движком баз данных **Atomic**.

### Синтаксис

```
RENAME DATABASE|TABLE|DICTIONARY name TO new_name [,...] [ON CLUSTER cluster]
```

## RENAME DATABASE

Переименовывает базы данных.

### Синтаксис

```
RENAME DATABASE atomic_database1 TO atomic_database2 [,...] [ON CLUSTER cluster]
```

# RENAME TABLE

Переименовывает одну или несколько таблиц.

Переименовывание таблиц является лёгкой операцией. Если вы указали после `TO` другую базу данных, то таблица будет перенесена в эту базу данных. При этом директории с базами данных должны быть расположены в одной файловой системе, иначе возвращается ошибка. Если переименовывается несколько таблиц в одном запросе, то такая операция неатомарная. Она может выполнится частично, и запросы в других сессиях могут получить ошибку `Table ... doesn't exist...`.

## Синтаксис

```
RENAME TABLE [db1.]name1 TO [db2.]name2 [...] [ON CLUSTER cluster]
```

## Пример

```
RENAME TABLE table_A TO table_A_bak, table_B TO table_B_bak;
```

# RENAME DICTIONARY

Переименовывает один или несколько словарей. Этот запрос можно использовать для перемещения словарей между базами данных.

## Синтаксис

```
RENAME DICTIONARY [db0.]dict_A TO [db1.]dict_B [...] [ON CLUSTER cluster]
```

## Смотрите также

- [Словари](#)

# EXCHANGE

Атомарно обменивает имена двух таблиц или словарей.

Это действие также можно выполнить с помощью запроса [RENAME](#), используя третье временное имя, но в таком случае действие неатомарно.

## Примечание

Запрос `EXCHANGE` поддерживается только движком баз данных [Atomic](#).

## Синтаксис

```
EXCHANGE TABLES|DICTIONARIES [db0.]name_A AND [db1.]name_B
```

# EXCHANGE TABLES

Обменивает имена двух таблиц.

## Синтаксис

```
EXCHANGE TABLES [db0.]table_A AND [db1.]table_B
```

## EXCHANGE DICTIONARIES

Обменивает имена двух словарей.

### Синтаксис

```
EXCHANGE DICTIONARIES [db0.]dict_A AND [db1.]dict_B
```

### Смотрите также

- [Словари](#)

## SET

```
SET param = value
```

Устанавливает значение `value` для [настройки](#) `param` в текущей сессии. [Конфигурационные параметры сервера](#) нельзя изменить подобным образом.

Можно одним запросом установить все настройки из заданного профиля настроек.

```
SET profile = 'profile-name-from-the-settings-file'
```

Подробности смотрите в разделе [Настройки](#).

## SET ROLE

Активирует роли для текущего пользователя.

### Синтаксис

```
SET ROLE {DEFAULT | NONE | role [...] | ALL | ALL EXCEPT role [...]}
```

## SET DEFAULT ROLE

Устанавливает роли по умолчанию для пользователя.

Роли по умолчанию активируются автоматически при входе пользователя. Ролями по умолчанию могут быть установлены только ранее назначенные роли. Если роль не назначена пользователю, ClickHouse выбрасывает исключение.

### Синтаксис

```
SET DEFAULT ROLE {NONE | role [...] | ALL | ALL EXCEPT role [...]} TO {user|CURRENT_USER} [...]
```

## Примеры

Установить несколько ролей по умолчанию для пользователя:

```
SET DEFAULT ROLE role1, role2, ... TO user
```

Установить ролями по умолчанию все назначенные пользователю роли:

```
SET DEFAULT ROLE ALL TO user
```

Удалить роли по умолчанию для пользователя:

```
SET DEFAULT ROLE NONE TO user
```

Установить ролями по умолчанию все назначенные пользователю роли за исключением указанных:

```
SET DEFAULT ROLE ALL EXCEPT role1, role2 TO user
```

## TRUNCATE

```
TRUNCATE TABLE [IF EXISTS] [db.]name [ON CLUSTER cluster]
```

Удаляет все данные из таблицы. Если условие IF EXISTS не указано, запрос вернет ошибку, если таблицы не существует.

Запрос TRUNCATE не поддерживается для следующих движков: [View](#), [File](#), [URL](#), [Buffer](#) и [Null](#).

Вы можете настроить ожидание выполнения действий на репликах с помощью настройки [replication\\_alter\\_partitions\\_sync](#).

Вы можете указать время ожидания (в секундах) выполнения запросов TRUNCATE для неактивных реплик с помощью настройки [replication\\_wait\\_for\\_inactive\\_replica\\_timeout](#).

### Примечание

Если значение настройки replication\_alter\_partitions\_sync равно 2 и некоторые реплики не активны больше времени, заданного настройкой replication\_wait\_for\_inactive\_replica\_timeout, то генерируется исключение UNFINISHED.

## USE

```
USE db
```

Позволяет установить текущую базу данных для сессии.

Текущая база данных используется для поиска таблиц, если база данных не указана в запросе явно через точку перед именем таблицы.

При использовании HTTP протокола запрос не может быть выполнен, так как понятия сессии не существует.

## Запрос WATCH

## Важно

Это экспериментальная функция. Она может повлечь потерю совместимости в будущих версиях.

Чтобы использовать `LIVE VIEW` и запросы `WATCH`, включите настройку `set allow_experimental_live_view = 1.`

### Синтаксис

```
WATCH [db.]live_view [EVENTS] [LIMIT n] [FORMAT format]
```

Запрос `WATCH` постоянно возвращает содержимое **LIVE-представления**. Если параметр `LIMIT` не был задан, запрос `WATCH` будет непрерывно обновлять содержимое **LIVE-представления**.

```
WATCH [db.]live_view;
```

## Виртуальные столбцы

Виртуальный столбец `_version` в результате запроса обозначает версию данного результата.

### Пример:

```
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

...

По умолчанию запрашиваемые данные возвращаются клиенту, однако в сочетании с запросом `INSERT INTO` они могут быть перенаправлены для вставки в другую таблицу.

### Пример:

```
INSERT INTO [db.]table WATCH [db.]live_view ...
```

## Секция EVENTS

С помощью параметра `EVENTS` можно получить компактную форму результата запроса `WATCH`. Вместо полного результата вы получаете номер последней версии результата.

```
WATCH [db.]live_view EVENTS;
```

### Пример:

```
CREATE LIVE VIEW lv WITH REFRESH 5 AS SELECT now();
WATCH lv EVENTS;
```

```
version
1 |
version
2 |
...
```

## Секция LIMIT

Параметр **LIMIT** n задает количество обновлений запроса **WATCH**, после которого отслеживание прекращается. По умолчанию это число не задано, поэтому запрос будет выполняться постоянно. Значение **LIMIT 0** означает, что запрос **WATCH** вернет единственный актуальный результат запроса и прекратит отслеживание.

```
WATCH [db.]live_view LIMIT 1;
```

### Пример:

```
CREATE LIVE VIEW lv WITH REFRESH 5 AS SELECT now();
WATCH lv EVENTS LIMIT 1;
```

```
version
1 |
```

## Секция FORMAT

Параметр **FORMAT** работает аналогично одноименному параметру запроса **SELECT**.

### Примечание

При отслеживании **LIVE VIEW** через интерфейс HTTP следует использовать формат **JSONEachRowWithProgress**. Постоянные сообщения об изменениях будут добавлены в поток вывода для поддержания активности долговременного HTTP-соединения до тех пор, пока результат запроса изменяется. Проомежуток времени между сообщениями об изменениях управляется настройкой **live\_view\_heartbeat\_interval**.

## Синтаксис

В системе есть два вида парсеров: полноценный парсер SQL (recursive descent parser) и парсер форматов данных (быстрый потоковый парсер).

Во всех случаях кроме запроса **INSERT**, используется только полноценный парсер SQL.

В запросе **INSERT** используется оба парсера:

```
INSERT INTO t VALUES (1, 'Hello, world'), (2, 'abc'), (3, 'def')
```

Фрагмент `INSERT INTO t VALUES` парсится полноценным парсером, а данные `(1, 'Hello, world'), (2, 'abc'), (3, 'def')` - быстрым потоковым парсером.

Данные могут иметь любой формат. При получении запроса, сервер заранее считывает в оперативку не более `max_query_size` байт запроса (по умолчанию, 1МБ), а всё остальное обрабатывается потоково.

Таким образом, в системе нет проблем с большими `INSERT` запросами, как в MySQL.

При использовании формата `Values` в `INSERT` запросе может сложиться иллюзия, что данные парсятся также, как выражения в запросе `SELECT`, но это не так. Формат `Values` гораздо более ограничен.

Далее пойдёт речь о полноценном парсере. О парсерах форматов, смотри раздел «Форматы».

## Пробелы

Между синтаксическими конструкциями (в том числе, в начале и конце запроса) может быть расположено произвольное количество пробельных символов. К пробельным символам относятся пробел, таб, перевод строки, CR, form feed.

## Комментарии

Поддерживаются комментарии в SQL-стиле и C-стиле.

Комментарии в SQL-стиле: от `--` до конца строки. Пробел после `--` может не ставиться.

Комментарии в C-стиле: от `/*` до `*/`. Такие комментарии могут быть многострочными. Пробелы тоже не обязательны.

## Ключевые слова

Ключевые слова не зависят от регистра, если они соответствуют:

- Стандарту SQL. Например, применение любого из вариантов `SELECT`, `select` или `SeLeCt` не вызовет ошибки.
- Реализации в некоторых популярных DBMS (MySQL или Postgres). Например, `DateTime` и `datetime`.

Зависимость от регистра для имён типов данных можно проверить в таблице `system.data_type_families`.

В отличие от стандарта SQL, все остальные ключевые слова, включая названия функций зависят от регистра.

Ключевые слова не зарезервированы (а всего лишь парсятся как ключевые слова в соответствующем контексте). Если вы используете [идентификаторы](#), совпадающие с ключевыми словами, заключите их в кавычки. Например, запрос `SELECT "FROM" FROM table_name` валиден, если таблица `table_name` имеет столбец с именем `"FROM"`.

## Идентификаторы

Идентификаторы:

- Имена кластеров, баз данных, таблиц, разделов и столбцов;
- Функции;
- Типы данных;
- [Синонимы выражений](#).

Некоторые идентификаторы нужно указывать в кавычках (например, идентификаторы с пробелами). Прочие идентификаторы можно указывать без кавычек. Рекомендуется использовать идентификаторы, не требующие кавычек.

Идентификаторы не требующие кавычек соответствуют регулярному выражению `^[a-zA-Z_][0-9a-zA-Z_]*$` и не могут совпадать с [ключевыми словами](#). Примеры: `x, _1, X_y_Z123_`.

Если вы хотите использовать идентификаторы, совпадающие с ключевыми словами, или использовать в идентификаторах символы, не входящие в регулярное выражение, заключите их в двойные или обратные кавычки, например, `"id"`, ``id``.

## Литералы

Существуют: числовые, строковые, составные литералы и `NULL`.

### Числовые

Числовой литерал пытается распарситься:

- Сначала как знаковое 64-разрядное число, функцией `strtoull`.
- Если не получилось, то как беззнаковое 64-разрядное число, функцией `strtol`.
- Если не получилось, то как число с плавающей запятой, функцией `strtod`.
- Иначе — ошибка.

Соответствующее значение будет иметь тип минимального размера, который вмещает значение. Например, `1` парсится как `UInt8`, а `256` как `UInt16`. Подробнее о типах данных читайте в разделе [Типы данных](#).

Примеры: `1, 18446744073709551615, 0xDEADBEEF, 01, 0.1, 1e100, -1e-100, inf, nan`.

### Строковые

Поддерживаются только строковые литералы в одинарных кавычках. Символы внутри могут быть экранированы с помощью обратного слеша. Следующие escape-последовательности имеют соответствующее специальное значение: `\b, \f, \r, \n, \t, \0, \a, \v, \xHH`. Во всех остальных случаях, последовательности вида `\c`, где `c` — любой символ, преобразуется в `c`. Таким образом, могут быть использованы последовательности `'\`` и `\```. Значение будет иметь тип [String](#).

Минимальный набор символов, которых вам необходимо экранировать в строковых литералах: `'` и `\``. Одинарная кавычка может быть экранирована одинарной кавычкой, литералы `'It\'s'` и `'It"s'` эквивалентны.

### Составные

Поддерживаются конструкции для массивов: `[1, 2, 3]` и кортежей: `(1, 'Hello, world!', 2)`.

На самом деле, это вовсе не литералы, а выражение с оператором создания массива и оператором создания кортежка, соответственно.

Массив должен состоять хотя бы из одного элемента, а кортеж - хотя бы из двух.

Кортежи носят служебное значение для использования в секции `IN` запроса `SELECT`. Кортежи могут быть получены как результат запроса, но они не могут быть сохранены в базе данных (за исключением таблицы [Memory](#).)

### NULL

Обозначает, что значение отсутствует.

Чтобы в поле таблицы можно было хранить `NULL`, оно должно быть типа [Nullable](#).

В зависимости от формата данных (входных или выходных) `NULL` может иметь различное представление. Подробнее смотрите в документации для [форматов данных](#).

При обработке `NULL` есть множество особенностей. Например, если хотя бы один из аргументов операции сравнения — `NULL`, то результатом такой операции тоже будет `NULL`. Этим же свойством обладают операции умножения, сложения и пр. Подробнее читайте в документации на каждую операцию.

В запросах можно проверить `NULL` с помощью операторов `IS NULL` и `IS NOT NULL`, а также соответствующих функций `isNull` и `isNotNull`.

## Heredoc

Синтаксис `heredoc` — это способ определения строк с сохранением исходного формата (часто с переносом строки). `Heredoc` задается как произвольный строковый литерал между двумя символами `$`, например `$heredoc$`. Значение между двумя `heredoc` обрабатывается "как есть".

Синтаксис `heredoc` часто используют для вставки кусков кода SQL, HTML, XML и т.п.

### Пример

Запрос:

```
SELECT $smth$SHOW CREATE VIEW my_view$smth$;
```

Результат:

```
'SHOW CREATE VIEW my_view'  
SHOW CREATE VIEW my_view |
```

## ФУНКЦИИ

Функции записываются как идентификатор со списком аргументов (возможно, пустым) в скобках. В отличие от стандартного SQL, даже в случае пустого списка аргументов, скобки обязательны. Пример: `now()`.

Бывают обычные и агрегатные функции (смотрите раздел «Агрегатные функции»). Некоторые агрегатные функции могут содержать два списка аргументов в круглых скобках. Пример: `quantile(0.9)(x)`. Такие агрегатные функции называются «параметрическими», а первый список аргументов называется «параметрами». Синтаксис агрегатных функций без параметров ничем не отличается от обычных функций.

## Операторы

Операторы преобразуются в соответствующие им функции во время парсинга запроса, с учётом их приоритета и ассоциативности.

Например, выражение `1 + 2 * 3 + 4` преобразуется в `plus(plus(1, multiply(2, 3)), 4)`.

## Типы данных и движки таблиц

Типы данных и движки таблиц в запросе `CREATE` записываются также, как идентификаторы или также как функции. То есть, могут содержать или не содержать список аргументов в круглых скобках. Подробнее смотрите разделы «Типы данных», «Движки таблиц», «CREATE».

## Синонимы выражений

Синоним — это пользовательское имя выражения в запросе.

## expr AS alias

- **AS** — ключевое слово для определения синонимов. Можно определить синоним для имени таблицы или столбца в секции `SELECT` без использования ключевого слова `AS`.

Например, `SELECT table\_name\_alias.column\_name FROM table\_name table\_name\_alias`.

В функции [CAST](#sql\_reference-syntax-md), ключевое слово `AS` имеет другое значение. Смотрите описание функции.

- **expr** — любое выражение, которое поддерживает ClickHouse.

Например, `SELECT column\_name \* 2 AS double FROM some\_table`.

- **alias** — имя для выражения. Синонимы должны соответствовать синтаксису [идентификаторов](#).

Например, `SELECT "table t".column\_name FROM table\_name AS "table t"`.

## Примечания по использованию

Синонимы являются глобальными для запроса или подзапроса, и вы можете определить синоним в любой части запроса для любого выражения. Например, `SELECT (1 AS n) + 2, n`

Синонимы не передаются в подзапросы и между подзапросами. Например, при выполнении запроса `SELECT (SELECT sum(b.a) + num FROM b) - a.a AS num FROM a` ClickHouse генерирует исключение `Unknown identifier: num`.

Если синоним определен для результирующих столбцов в секции `SELECT` вложенного запроса, то эти столбцы отображаются во внешнем запросе. Например, `SELECT n + m FROM (SELECT 1 AS n, 2 AS m)`

Будьте осторожны с синонимами, совпадающими с именами столбцов или таблиц. Рассмотрим следующий пример:

```
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.

В этом примере мы объявили таблицу `t` со столбцом `b`. Затем, при выборе данных, мы определили синоним `sum(b) AS b`. Поскольку синонимы глобальные, то ClickHouse заменил литерал `b` в выражении `argMax(a, b)` выражением `sum(b)`. Эта замена вызвала исключение. Можно изменить это поведение, включив настройку `prefer_column_name_to_alias`, для этого нужно установить ее в значение 1.

## Звёздочка

В запросе `SELECT`, вместо выражения может стоять звёздочка. Подробнее смотрите раздел «`SELECT`».

## Выражения

Выражение представляет собой функцию, идентификатор, литерал, применение оператора, выражение в скобках, подзапрос, звёздочку. А также может содержать синоним.

Список выражений - одно выражение или несколько выражений через запятую.

Функции и операторы, в свою очередь, в качестве аргументов, могут иметь произвольные выражения.

## Распределенные DDL запросы (секция ON CLUSTER)

Запросы `CREATE`, `DROP`, `ALTER`, `RENAME` поддерживают возможность распределенного выполнения на кластере.

Например, следующий запрос создает распределенную (Distributed) таблицу `all_hits` на каждом хосте в `cluster`:

```
CREATE TABLE IF NOT EXISTS all_hits ON CLUSTER cluster (p Date, i Int32) ENGINE = Distributed(cluster, default, hits)
```

Для корректного выполнения таких запросов необходимо на каждом хосте иметь одинаковое определение кластера (для упрощения синхронизации конфигов можете использовать подстановки из ZooKeeper). Также необходимо подключение к ZooKeeper серверам.

Локальная версия запроса в конечном итоге будет выполнена на каждом хосте кластера, даже если некоторые хосты в данный момент не доступны. Гарантируется упорядоченность выполнения запросов в рамках одного хоста.

## ФУНКЦИИ

Функции бывают как минимум\* двух видов - обычные функции (называются просто, функциями) и агрегатные функции. Это совершенно разные вещи. Обычные функции работают так, как будто применяются к каждой строке по отдельности (для каждой строки, результат вычисления функции не зависит от других строк). Агрегатные функции аккумулируют множество значений из разных строк (то есть, зависят от целого множества строк).

В этом разделе речь пойдёт об обычных функциях. Для агрегатных функций, смотрите раздел «Агрегатные функции».

\* - есть ещё третий вид функций, к которым относится функция `arrayJoin`; также можно отдельно иметь ввиду табличные функции.\*

## Строгая типизация

В ClickHouse, в отличие от стандартного SQL, типизация является строгой. То есть, не производится неявных преобразований между типами. Все функции работают для определённого набора типов. Это значит, что иногда вам придётся использовать функции преобразования типов.

## Склейка одинаковых выражений

Все выражения в запросе, имеющие одинаковые AST (одинаковую запись или одинаковый результат синтаксического разбора), считаются имеющими одинаковые значения. Такие выражения склеиваются и исполняются один раз. Одинаковые подзапросы тоже склеиваются.

# Типы результата

Все функции возвращают одно (не несколько, не ноль) значение в качестве результата. Тип результата обычно определяется только типами аргументов, но не значениями аргументов. Исключение - функция `tupleElement` (оператор `a.N`), а также функция `toFixedString`.

## Константы

Для простоты, некоторые функции могут работать только с константами в качестве некоторых аргументов. Например, правый аргумент оператора `LIKE` должен быть константой.

Почти все функции возвращают константу для константных аргументов. Исключение - функции генерации случайных чисел.

Функция `now` возвращает разные значения для запросов, выполненных в разное время, но результат считается константой, так как константность важна лишь в пределах одного запроса. Константное выражение также считается константой (например, правую часть оператора `LIKE` можно сконструировать из нескольких констант).

Функции могут быть по-разному реализованы для константных и не константных аргументов (выполняется разный код). Но результат работы для константы и полноценного столбца, содержащего только одно такое же значение, должен совпадать.

## Обработка NULL

Функции имеют следующие виды поведения:

- Если хотя бы один из аргументов функции — `NULL`, то результат функции тоже `NULL`.
- Специальное поведение, указанное в описании каждой функции отдельно. В исходном коде ClickHouse такие функции можно определить по свойству `UseDefaultImplementationForNulls=false`.

## Неизменяемость

Функции не могут поменять значения своих аргументов - любые изменения возвращаются в качестве результата. Соответственно, от порядка записи функций в запросе, результат вычислений отдельных функций не зависит.

## Функции высшего порядка, оператор `->` и функция `lambda(params, expr)`

Функции высшего порядка, в качестве своего функционального аргумента могут принимать только лямбда-функции. Чтобы передать лямбда-функцию в функцию высшего порядка, используйте оператор `->`. Слева от стрелочки стоит формальный параметр — произвольный идентификатор, или несколько формальных параметров — произвольные идентификаторы в кортеже. Справа от стрелочки стоит выражение, в котором могут использоваться эти формальные параметры, а также любые столбцы таблицы.

Примеры:

```
x -> 2 * x
str -> str != Referer
```

В функции высшего порядка может быть передана лямбда-функция, принимающая несколько аргументов. В этом случае в функцию высшего порядка передаётся несколько массивов одинаковой длины, которым эти аргументы будут соответствовать.

Для некоторых функций первый аргумент (лямбда-функция) может отсутствовать. В этом случае подразумевается тождественное отображение.

# Пользовательские функции SQL

Функции можно создавать из лямбда выражений с помощью **CREATE FUNCTION**. Для удаления таких функций используется выражение **DROP FUNCTION**.

## Исполняемые пользовательские функции

ClickHouse может вызывать внешнюю программу или скрипт для обработки данных. Такие функции описываются в **конфигурационном файле**. Путь к нему должен быть указан в настройке `user_defined_executable_functions_config` в основной конфигурации. В пути можно использовать символ подстановки `*`, тогда будут загружены все файлы, соответствующие шаблону. Пример:

```
<user_defined_executable_functions_config>*_function.xml</user_defined_executable_functions_config>
```

Файлы с описанием функций ищутся относительно каталога, заданного в настройке `user_files_path`.

Конфигурация функции содержит следующие настройки:

- `name` - имя функции.
- `command` - исполняемая команда или скрипт.
- `argument` - описание аргумента, содержащее его тип во вложенной настройке `type`. Каждый аргумент описывается отдельно.
- `format` - **формат** передачи аргументов.
- `return_type` - тип возвращаемого значения.
- `type` - вариант запуска команды. Если задан вариант `executable`, то запускается одна команда. При указании `executable_pool` создается пул команд. Необязательная настройка. Значение по умолчанию `10`.
- `max_command_execution_time` - максимальное время в секундах, которое отводится на обработку блока данных. Эта настройка применима только для команд с вариантом запуска `executable_pool`. Необязательная настройка. Значение по умолчанию `10`.
- `command_termination_timeout` - максимальное время завершения команды в секундах после закрытия конвейера. Если команда не завершается, то процессу отправляется сигнал `SIGTERM`. Эта настройка применима только для команд с вариантом запуска `executable_pool`. Необязательная настройка. Значение по умолчанию `10`.
- `pool_size` - размер пула команд. Необязательная настройка. Значение по умолчанию `16`.
- `lifetime` - интервал перезагрузки функций в секундах. Если задан `0`, то функция не перезагружается.
- `send_chunk_header` - управляет отправкой количества строк перед отправкой блока данных для обработки. Необязательная настройка. Значение по умолчанию `false`.

Команда должна читать аргументы из `STDIN` и выводить результат в `STDOUT`. Обработка должна выполняться в цикле. То есть после обработки группы аргументов команда должна ожидать следующую группу.

### Пример

XML конфигурация, описывающая функцию `test_function`:

```
<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>
```

Запрос:

```
SELECT test_function(toUInt64(2), toUInt64(2));
```

Результат:

```
└─test_function(toUInt64(2), toUInt64(2))─
   4 |
```

## Обработка ошибок

Некоторые функции могут кидать исключения в случае ошибочных данных. В этом случае, выполнение запроса прерывается, и текст ошибки выводится клиенту. При распределённой обработке запроса, при возникновении исключения на одном из серверов, на другие серверы пытается отправиться просьба тоже прервать выполнение запроса.

## Вычисление выражений-аргументов

В почти всех языках программирования, для некоторых операторов может не вычисляться один из аргументов. Обычно - для операторов `&&`, `||`, `?:`.

Но в ClickHouse, аргументы функций (операторов) вычисляются всегда. Это связано с тем, что вычисления производятся не по отдельности для каждой строки, а сразу для целых кусочков столбцов.

## Выполнение функций при распределённой обработке запроса

При распределённой обработке запроса, как можно большая часть стадий выполнения запроса производится на удалённых серверах, а оставшиеся стадии (слияние промежуточных результатов и всё, что дальше) - на сервере-инициаторе запроса.

Это значит, что выполнение функций может производиться на разных серверах.

Например, в запросе `SELECT f(sum(g(x))) FROM distributed_table GROUP BY h(y)`,

- если `distributed_table` имеет хотя бы два шарда, то функции `g` и `h` выполняются на удалённых серверах, а функция `f` - на сервере-инициаторе запроса;
- если `distributed_table` имеет только один шард, то все функции `f`, `g`, `h` выполняются на сервере этого шарда.

Обычно результат выполнения функции не зависит от того, на каком сервере её выполнить. Но иногда это довольно важно.

Например, функции, работающие со словарями, будут использовать словарь, присутствующий на том сервере, на котором они выполняются.

Другой пример - функция `hostName` вернёт имя сервера, на котором она выполняется, и это можно использовать для служебных целей - чтобы в запросе `SELECT` сделать `GROUP BY` по серверам.

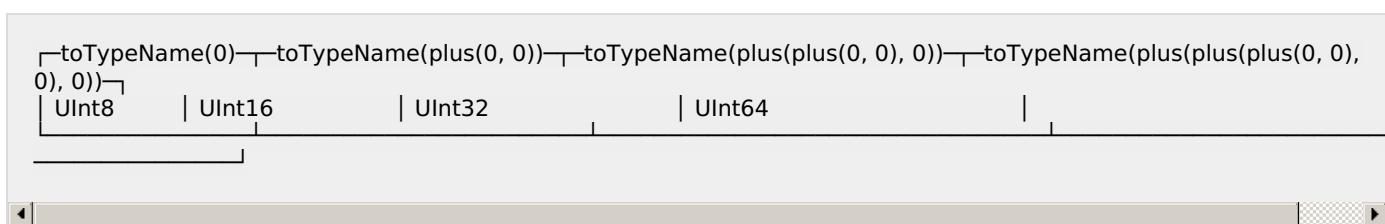
Если функция в запросе выполняется на сервере-инициаторе запроса, а вам нужно, чтобы она выполнялась на удалённых серверах, вы можете обернуть её в агрегатную функцию `any` или добавить в ключ в `GROUP BY`.

## Арифметические функции

Для всех арифметических функций, тип результата вычисляется, как минимальный числовой тип, который может вместить результат, если такой тип есть. Минимум берётся одновременно по числу бит, знаковости и «плавучести». Если бит не хватает, то берётся тип максимальной битности.

Пример:

```
SELECT toTypeName(0), toTypeName(0 + 0), toTypeName(0 + 0 + 0), toTypeName(0 + 0 + 0 + 0)
```



Арифметические функции работают для любой пары типов из `UInt8`, `UInt16`, `UInt32`, `UInt64`, `Int8`, `Int16`, `Int32`, `Int64`, `Float32`, `Float64`.

Переполнение производится также, как в C++.

### `plus(a, b)`, оператор `a + b`

Вычисляет сумму чисел.

Также можно складывать целые числа с датой и датой-с-временем. В случае даты, прибавление целого числа означает прибавление соответствующего количества дней. В случае даты-с-временем - прибавление соответствующего количества секунд.

### `minus(a, b)`, оператор `a - b`

Вычисляет разность чисел. Результат всегда имеет знаковый тип.

Также можно вычитать целые числа из даты и даты-с-временем. Смысл аналогичен -смотрите выше для `plus`.

### `multiply(a, b)`, оператор `a * b`

Вычисляет произведение чисел.

### `divide(a, b)`, оператор `a / b`

Вычисляет частное чисел. Тип результата всегда является типом с плавающей запятой.

То есть, деление не целочисленное. Для целочисленного деления, используйте функцию `intDiv`. При делении на ноль получится `inf`, `-inf` или `nan`.

## **intDiv(a, b)**

Вычисляет частное чисел. Деление целочисленное, с округлением вниз (по абсолютному значению). При делении на ноль или при делении минимального отрицательного числа на минус единицу, кидается исключение.

## **intDivOrZero(a, b)**

Отличается от `intDiv` тем, что при делении на ноль или при делении минимального отрицательного числа на минус единицу, возвращается ноль.

## **modulo(a, b), оператор a % b**

Вычисляет остаток от деления.

Если аргументы - числа с плавающей запятой, то они предварительно преобразуются в целые числа, путём отбрасывания дробной части.

Берётся остаток в том же смысле, как это делается в C++. По факту, для отрицательных чисел, используется truncated division.

При делении на ноль или при делении минимального отрицательного числа на минус единицу, кидается исключение.

## **moduloOrZero(a, b)**

В отличие от `modulo`, возвращает ноль при делении на ноль.

## **negate(a), оператор -a**

Вычисляет число, обратное по знаку. Результат всегда имеет знаковый тип.

## **abs(a)**

Вычисляет абсолютное значение для числа a. То есть, если  $a < 0$ , то возвращает  $-a$ .

Для беззнаковых типов ничего не делает. Для чисел типа целых со знаком, возвращает число беззнакового типа.

## **gcd(a, b)**

Вычисляет наибольший общий делитель чисел.

При делении на ноль или при делении минимального отрицательного числа на минус единицу, кидается исключение.

## **lcm(a, b)**

Вычисляет наименьшее общее кратное чисел.

При делении на ноль или при делении минимального отрицательного числа на минус единицу, кидается исключение.

## **max2**

Сравнивает два числа и возвращает максимум. Возвращаемое значение приводится к типу `Float64`.

### **Синтаксис**

```
max2(value1, value2)
```

### **Аргументы**

- `value1` — первое число. [Int/UInt](#) или [Float](#).
- `value2` — второе число. [Int/UInt](#) или [Float](#).

### Возвращаемое значение

- Максимальное значение среди двух чисел.

Тип: [Float](#).

### Пример

Запрос:

```
SELECT max2(-1, 2);
```

Результат:

```
max2(-1, 2)
  2 |
```

## min2

Сравнивает два числа и возвращает минимум. Возвращаемое значение приводится к типу [Float64](#).

### Синтаксис

```
min2(value1, value2)
```

### Аргументы

- `value1` — первое число. [Int/UInt](#) или [Float](#).
- `value2` — второе число. [Int/UInt](#) или [Float](#).

### Возвращаемое значение

- Минимальное значение среди двух чисел.

Тип: [Float](#).

### Пример

Запрос:

```
SELECT min2(-1, 2);
```

Результат:

```
min2(-1, 2)
 -1 |
```

## Массивы

# empty

Проверяет, является ли входной массив пустым.

## Синтаксис

```
empty([x])
```

Массив считается пустым, если он не содержит ни одного элемента.

## Примечание

Функцию можно оптимизировать, если включить настройку

**optimize\_functions\_to\_subcolumns**. При `optimize_functions_to_subcolumns = 1` функция читает только подстолбец `size0` вместо чтения и обработки всего столбца массива. Запрос `SELECT empty(arr) FROM TABLE` преобразуется к запросу `SELECT arr.size0 = 0 FROM TABLE`.

Функция также поддерживает работу с типами `String` и `UUID`.

## Параметры

- [x] — массив на входе функции. `Array`.

## Возвращаемое значение

- Возвращает 1 для пустого массива или 0 — для непустого массива.

Тип: `UInt8`.

## Пример

Запрос:

```
SELECT empty([]);
```

Ответ:

```
empty(array())  
1 |
```

# notEmpty

Проверяет, является ли входной массив непустым.

## Синтаксис

```
notEmpty([x])
```

Массив считается непустым, если он содержит хотя бы один элемент.

## Примечание

Функцию можно оптимизировать, если включить настройку **optimize\_functions\_to\_subcolumns**. При `optimize_functions_to_subcolumns = 1` функция читает только подстолбец **size0** вместо чтения и обработки всего столбца массива. Запрос `SELECT notEmpty(arr) FROM table` преобразуется к запросу `SELECT arr.size0 != 0 FROM TABLE`.

Функция также поддерживает работу с типами **String** и **UUID**.

## Параметры

- [x] — массив на входе функции. **Array**.

## Возвращаемое значение

- Возвращает 1 для непустого массива или 0 — для пустого массива.

Тип: **UInt8**.

## Пример

Запрос:

```
SELECT notEmpty([1,2]);
```

Результат:

```
+-----+  
| notEmpty([1, 2]) |  
+-----+  
| 1 |
```

## length

Возвращает количество элементов в массиве.

Тип результата - **UInt64**.

Функция также работает для строк.

Функцию можно оптимизировать, если включить настройку **optimize\_functions\_to\_subcolumns**. При `optimize_functions_to_subcolumns = 1` функция читает только подстолбец **size0** вместо чтения и обработки всего столбца массива. Запрос `SELECT length(arr) FROM table` преобразуется к запросу `SELECT arr.size0 FROM TABLE`.

**emptyArrayUInt8, emptyArrayUInt16, emptyArrayUInt32, emptyArrayUInt64**

**emptyArrayInt8, emptyArrayInt16, emptyArrayInt32, emptyArrayInt64**

**emptyArrayFloat32, emptyArrayFloat64**

**emptyArrayDate, emptyArrayDateTime**

**emptyArrayString**

Принимает ноль аргументов и возвращает пустой массив соответствующего типа.

**emptyArrayToSingle**

Принимает пустой массив и возвращает массив из одного элемента, равного значению по умолчанию.

## range(end), range([start, ] end [, step])

Возвращает массив чисел от `start` до `end - 1` с шагом `step`.

### Синтаксис

```
range([start, ] end [, step])
```

### Аргументы

- `start` — начало диапазона. Обязательно, когда указан `step`. По умолчанию равно 0. Тип: `UInt`
- `end` — конец диапазона. Обязательный аргумент. Должен быть больше, чем `start`. Тип: `UInt`
- `step` — шаг обхода. Необязательный аргумент. По умолчанию равен 1. Тип: `UInt`

### Возвращаемые значения

- массив `UInt` чисел от `start` до `end - 1` с шагом `step`

### Особенности реализации

- Не поддерживаются отрицательные значения аргументов: `start`, `end`, `step` имеют тип `UInt`.
- Если в результате запроса создаются массивы суммарной длиной больше, чем количество элементов, указанное настройкой `function_range_max_elements_in_block`, то генерируется исключение.

### Примеры

Запрос:

```
SELECT range(5), range(1, 5), range(1, 5, 2);
```

Ответ:

```
range(5)———range(1, 5)——range(1, 5, 2)——  
[0,1,2,3,4] | [1,2,3,4] | [1,3] |
```

## array(x1, ...), оператор [x1, ...]

Создаёт массив из аргументов функции.

Аргументы должны быть константами и иметь типы, для которых есть наименьший общий тип. Должен быть передан хотя бы один аргумент, так как иначе непонятно, какого типа создавать массив. То есть, с помощью этой функции невозможно создать пустой массив (для этого используйте функции `emptyArray*`, описанные выше).

Возвращает результат типа `Array(T)`, где `T` - наименьший общий тип от переданных аргументов.

## arrayConcat

Объединяет массивы, переданные в качестве аргументов.

```
arrayConcat(arrays)
```

## Аргументы

- `arrays` – произвольное количество элементов типа **Array**

### Пример

```
SELECT arrayConcat([1, 2], [3, 4], [5, 6]) AS res
```

```
res  
[1,2,3,4,5,6] |
```

## arrayElement(arr, n), operator arr[n]

Достаёт элемент с индексом `n` из массива `arr`. `n` должен быть любым целочисленным типом. Индексы в массиве начинаются с единицы.

Поддерживаются отрицательные индексы. В этом случае, будет выбран соответствующий по номеру элемент с конца. Например, `arr[-1]` - последний элемент массива.

Если индекс выходит за границы массива, то возвращается некоторое значение по умолчанию (0 для чисел, пустая строка для строк и т. п.), кроме случая с неконстантным массивом и константным индексом 0 (в этом случае будет ошибка `Array indices are 1-based`).

## has(arr, elem)

Проверяет наличие элемента `elem` в массиве `arr`.

Возвращает 0, если элемента в массиве нет, или 1, если есть.

`NULL` обрабатывается как значение.

```
SELECT has([1, 2, NULL], NULL)
```

```
has([1, 2, NULL], NULL)  
1 |
```

## hasAll

Проверяет, является ли один массив подмножеством другого.

```
hasAll(set, subset)
```

## Аргументы

- `set` – массив любого типа с набором элементов.
- `subset` – массив любого типа со значениями, которые проверяются на вхождение в `set`.

## Возвращаемые значения

- 1, если `set` содержит все элементы из `subset`.
- 0, в противном случае.

## Особенности

- Пустой массив является подмножеством любого массива.

- `NULL` обрабатывается как значение.
- Порядок значений в обоих массивах не имеет значения.

## Примеры

`SELECT hasAll([], [])` возвращает 1.

`SELECT hasAll([1, Null], [Null])` возвращает 1.

`SELECT hasAll([1.0, 2, 3, 4], [1, 3])` возвращает 1.

`SELECT hasAll(['a', 'b'], ['a'])` возвращает 1.

`SELECT hasAll([1], ['a'])` возвращает 0.

`SELECT hasAll([[1, 2], [3, 4]], [[1, 2], [3, 5]])` возвращает 0.

## hasAny

Проверяет, имеют ли два массива хотя бы один общий элемент.

```
hasAny(array1, array2)
```

## Аргументы

- `array1` – массив любого типа с набором элементов.
- `array2` – массив любого типа с набором элементов.

## Возвращаемые значения

- 1, если `array1` и `array2` имеют хотя бы один одинаковый элемент.
- 0, в противном случае.

## Особенности

- `NULL` обрабатывается как значение.
- Порядок значений в обоих массивах не имеет значения.

## Примеры

`SELECT hasAny([1], [])` возвращает 0.

`SELECT hasAny([Null], [Null, 1])` возвращает 1.

`SELECT hasAny([-128, 1., 512], [1])` возвращает 1.

`SELECT hasAny([[1, 2], [3, 4]], ['a', 'c'])` возвращает 0.

`SELECT hasAll([[1, 2], [3, 4]], [[1, 2], [1, 2]])` возвращает 1.

## indexOf(arr, x)

Возвращает индекс первого элемента `x` (начиная с 1), если он есть в массиве, или 0, если его нет.

Пример:

```
SELECT indexOf([1, 3, NULL, NULL], NULL)
```

```
indexOf([1, 3, NULL, NULL], NULL)─  
      3 |
```

Элементы, равные `NULL`, обрабатываются как обычные значения.

## arrayCount([func,] arr1, ...)

Возвращает количество элементов массива `arr`, для которых функция `func` возвращает не 0. Если `func` не указана - возвращает количество ненулевых элементов массива.

Функция `arrayCount` является **функцией высшего порядка** — в качестве первого аргумента ей можно передать лямбда-функцию.

## countEqual(arr, x)

Возвращает количество элементов массива, равных `x`. Эквивалентно `arrayCount(elem -> elem = x, arr)`.

`NULL` обрабатывается как значение.

Пример:

```
SELECT countEqual([1, 2, NULL, NULL], NULL)
```

```
countEqual([1, 2, NULL, NULL], NULL)─  
      2 |
```

## arrayEnumerate(arr)

Возвращает массив `[1, 2, 3, ..., length(arr)]`

Эта функция обычно используется совместно с `ARRAY JOIN`. Она позволяет, после применения `ARRAY JOIN`, посчитать что-либо только один раз для каждого массива. Пример:

```
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 |
```

В этом примере, `Reaches` - число достижений целей (строк, получившихся после применения `ARRAY JOIN`), а `Hits` - число хитов (строк, которые были до `ARRAY JOIN`). В данном случае, тот же результат можно получить проще:

```

SELECT
    sum(length(GoalsReached)) AS Reaches,
    count() AS Hits
FROM test.hits
WHERE (CounterID = 160656) AND notEmpty(GoalsReached)

```

Reaches	Hits
95606	31406

Также эта функция может быть использована в функциях высшего порядка. Например, с её помощью можно достать индексы массива для элементов, удовлетворяющих некоторому условию.

## arrayEnumerateUniq(arr, ...)

Возвращает массив, такого же размера, как исходный, где для каждого элемента указано, какой он по счету среди элементов с таким же значением.

Например: `arrayEnumerateUniq([10, 20, 10, 30]) = [1, 1, 2, 1].`

Эта функция полезна при использовании ARRAY JOIN и агрегации по элементам массива.

Пример:

```

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

В этом примере, для каждого идентификатора цели, посчитано количество достижений целей (каждый элемент вложенной структуры данных Goals является достижением целей) и количество визитов. Если бы не было ARRAY JOIN, мы бы считали количество визитов как `sum(Sign)`. Но в данном случае, строчки были размножены по вложенной структуре Goals, и чтобы после этого учесть каждый визит один раз, мы поставили условие на значение функции `arrayEnumerateUniq(Goals.ID)`.

Функция `arrayEnumerateUniq` может принимать несколько аргументов - массивов одинаковых размеров. В этом случае, уникальность считается для кортежей элементов на одинаковых позициях всех массивов.

```

SELECT arrayEnumerateUniq([1, 1, 1, 2, 2, 2], [1, 1, 2, 1, 1, 2]) AS res

```

```
res  
[1,2,1,1,2,1] |
```

Это нужно при использовании ARRAY JOIN с вложенной структурой данных и затем агрегации по нескольким элементам этой структуры.

## arrayPopBack

Удаляет последний элемент из массива.

```
arrayPopBack(array)
```

### Аргументы

- `array` – массив.

### Пример

```
SELECT arrayPopBack([1, 2, 3]) AS res;
```

```
res  
[1,2] |
```

## arrayPopFront

Удаляет первый элемент из массива.

```
arrayPopFront(array)
```

### Аргументы

- `array` – массив.

### Пример

```
SELECT arrayPopFront([1, 2, 3]) AS res;
```

```
res  
[2,3] |
```

## arrayPushBack

Добавляет один элемент в конец массива.

```
arrayPushBack(array, single_value)
```

### Аргументы

- `array` – массив.

- `single_value` – значение добавляемого элемента. В массив с числами можно добавить только числа, в массив со строками только строки. При добавлении чисел ClickHouse автоматически приводит тип `single_value` к типу данных массива. Подробнее о типах данных в ClickHouse читайте в разделе «[Типы данных](#)». Может быть равно `NULL`, в этом случае функция добавит элемент `NULL` в массив, а тип элементов массива преобразует в `Nullable`.

## Пример

```
SELECT arrayPushBack(['a'], 'b') AS res;
```

res

['a','b'] |

## arrayPushFront

Добавляет один элемент в начало массива.

```
arrayPushFront(array, single_value)
```

## Аргументы

- `array` – массив.
- `single_value` – значение добавляемого элемента. В массив с числами можно добавить только числа, в массив со строками только строки. При добавлении чисел ClickHouse автоматически приводит тип `single_value` к типу данных массива. Подробнее о типах данных в ClickHouse читайте в разделе «[Типы данных](#)». Может быть равно `NULL`, в этом случае функция добавит элемент `NULL` в массив, а тип элементов массива преобразует в `Nullable`.

## Пример

```
SELECT arrayPushFront(['b'], 'a') AS res;
```

res

['a','b'] |

## arrayResize

Изменяет длину массива.

```
arrayResize(array, size[, extender])
```

## Аргументы

- `array` — массив.
- `size` — необходимая длина массива.
  - Если `size` меньше изначального размера массива, то массив обрезается справа.
  - Если `size` больше изначального размера массива, массив дополняется справа значениями `extender` или значениями по умолчанию для типа данных элементов массива.
- `extender` — значение для дополнения массива. Может быть `NULL`.

## **Возвращаемое значение:**

Массив длины `size`.

## **Примеры вызовов**

```
SELECT arrayResize([1], 3);
```

```
arrayResize([1], 3)─  
[1,0,0] |
```

```
SELECT arrayResize([1], 3, NULL);
```

```
arrayResize([1], 3, NULL)─  
[1,NULL,NULL] |
```

## **arraySlice**

Возвращает срез массива.

```
arraySlice(array, offset[, length])
```

### **Аргументы**

- `array` – массив данных.
- `offset` – отступ от края массива. Положительное значение - отступ слева, отрицательное значение - отступ справа. Отсчет элементов массива начинается с 1.
- `length` – длина необходимого среза. Если указать отрицательное значение, то функция вернёт открытый срез `[offset, array_length - length]`. Если не указать значение, то функция вернёт срез `[offset, the_end_of_array]`.

## **Пример**

```
SELECT arraySlice([1, 2, NULL, 4, 5], 2, 3) AS res;
```

```
res─  
[2,NULL,4] |
```

Элементы массива равные `NULL` обрабатываются как обычные значения.

## **arraySort([func,] arr, ...)**

Возвращает массив `arr`, отсортированный в восходящем порядке. Если задана функция `func`, то порядок сортировки определяется результатом применения этой функции на элементы массива `arr`. Если `func` принимает несколько аргументов, то в функцию `arraySort` нужно передавать несколько массивов, которые будут соответствовать аргументам функции `func`. Подробные примеры рассмотрены в конце описания `arraySort`.

Пример сортировки целочисленных значений:

```
SELECT arraySort([1, 3, 3, 0])
```

```
arraySort([1, 3, 3, 0])—  
[0,1,3,3] |
```

Пример сортировки строковых значений:

```
SELECT arraySort(['hello', 'world', '!'])
```

```
arraySort(['hello', 'world', '!'])—  
['!', 'hello', 'world'] |
```

Значения `NULL`, `NaN` и `Inf` сортируются по следующему принципу:

```
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] |
```

- Значения `-Inf` идут в начале массива.
- Значения `NULL` идут в конце массива.
- Значения `NaN` идут перед `NULL`.
- Значения `Inf` идут перед `NaN`.

Функция `arraySort` является **функцией высшего порядка** — в качестве первого аргумента ей можно передать лямбда-функцию. В этом случае порядок сортировки определяется результатом применения лямбда-функции на элементы массива.

Рассмотрим пример:

```
SELECT arraySort((x) -> -x, [1, 2, 3]) as res;
```

```
res—  
[3,2,1] |
```

Для каждого элемента исходного массива лямбда-функция возвращает ключ сортировки, то есть  $[1 \rightarrow -1, 2 \rightarrow -2, 3 \rightarrow -3]$ . Так как `arraySort` сортирует элементы в порядке возрастания ключей, результат будет  $[3, 2, 1]$ . Как можно заметить, функция  $x \rightarrow -x$  устанавливает **обратный порядок сортировки**.

Лямбда-функция может принимать несколько аргументов. В этом случае, в функцию `arraySort` нужно передавать несколько массивов, которые будут соответствовать аргументам лямбда-функции (массивы должны быть одинаковой длины). Следует иметь в виду, что результат будет содержать элементы только из первого массива; элементы из всех последующих массивов будут задавать ключи сортировки. Например:

```
SELECT arraySort((x, y) -> y, ['hello', 'world'], [2, 1]) as res;
```

```
res  
['world', 'hello'] |
```

Элементы, указанные во втором массиве ([2,1]), определяют ключ сортировки для элементов из исходного массива ([‘hello’, ‘world’]), то есть [‘hello’ -> 2, ‘world’ -> 1]. Так как лямбда-функция не использует `x`, элементы исходного массива не влияют на порядок сортировки. Таким образом, ‘hello’ будет вторым элементом в отсортированном массиве, а ‘world’ — первым.

Ниже приведены другие примеры.

```
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] |
```

## Примечание

Для улучшения эффективности сортировки применяется [преобразование Шварца](#).

## arrayReverseSort([func,] arr, ...)

Возвращает массив `arr`, отсортированный в нисходящем порядке. Если указана функция `func`, то массив `arr` сначала сортируется в порядке, который определяется функцией `func`, а затем отсортированный массив переворачивается. Если функция `func` принимает несколько аргументов, то в функцию `arrayReverseSort` необходимо передавать несколько массивов, которые будут соответствовать аргументам функции `func`. Подробные примеры рассмотрены в конце описания функции `arrayReverseSort`.

Пример сортировки целочисленных значений:

```
SELECT arrayReverseSort([1, 3, 3, 0]);
```

```
arrayReverseSort([1, 3, 3, 0])  
[3,3,1,0] |
```

Пример сортировки строковых значений:

```
SELECT arrayReverseSort(['hello', 'world', '!']);
```

```
arrayReverseSort(['hello', 'world', '!']) →  
['world', 'hello', '!']
```

Значения `NULL`, `NaN` и `Inf` сортируются в следующем порядке:

```
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]
```

- Значения `Inf` идут в начале массива.
- Значения `NULL` идут в конце массива.
- Значения `NaN` идут перед `NULL`.
- Значения `-Inf` идут перед `NaN`.

Функция `arrayReverseSort` является **функцией высшего порядка** — в качестве первого аргумента ей можно передать лямбда-функцию. Например:

```
SELECT arrayReverseSort((x) -> -x, [1, 2, 3]) as res;
```

```
res →  
[1, 2, 3]
```

В этом примере, порядок сортировки устанавливается следующим образом:

1. Сначала исходный массив (`[1, 2, 3]`) сортируется в том порядке, который определяется лямбда-функцией. Результатом будет массив `[3, 2, 1]`.
2. Массив, который был получен на предыдущем шаге, переворачивается. То есть, получается массив `[1, 2, 3]`.

Лямбда-функция может принимать на вход несколько аргументов. В этом случае, в функцию `arrayReverseSort` нужно передавать несколько массивов, которые будут соответствовать аргументам лямбда-функции (массивы должны быть одинаковой длины). Следует иметь в виду, что результат будет содержать элементы только из первого массива; элементы из всех последующих массивов будут определять ключи сортировки. Например:

```
SELECT arrayReverseSort((x, y) -> y, ['hello', 'world'], [2, 1]) as res;
```

```
res →  
['hello', 'world']
```

В этом примере, массив сортируется следующим образом:

1. Сначала массив сортируется в том порядке, который определяется лямбда-функцией. Элементы, указанные во втором массиве (`[2, 1]`), определяют ключи сортировки соответствующих элементов из исходного массива (`['hello', 'world']`). То есть, будет массив `['world', 'hello']`.

2. Массив, который был отсортирован на предыдущем шаге, переворачивается. Получается массив ['hello', 'world'].

Ниже приведены ещё примеры.

```
SELECT arrayReverseSort((x, y) -> y, [0, 1, 2], ['c', 'b', 'a']) as res;
```

```
res  
[0,1,2] |
```

```
SELECT arrayReverseSort((x, y) -> -y, [4, 3, 5], [1, 2, 3]) AS res;
```

```
res  
[4,3,5] |
```

## arrayUniq(arr, ...)

Если передан один аргумент, считает количество разных элементов в массиве.

Если передано несколько аргументов, считает количество разных кортежей из элементов на соответствующих позициях в нескольких массивах.

Если необходимо получить список уникальных элементов массива, можно воспользоваться `arrayReduce('groupUniqArray', arr)`.

## arrayJoin(arr)

Особенная функция. Смотрите раздел [«Функция arrayJoin»](#).

## arrayDifference

Вычисляет разность между соседними элементами массива. Возвращает массив, где первым элементом будет 0, вторым – разность `a[1] - a[0]` и т. д. Тип элементов результирующего массива определяется правилами вывода типов при вычитании (напр. `UInt8 - UInt8 = Int16`).

### Синтаксис

```
arrayDifference(array)
```

### Аргументы

- `array` – массив.

### Возвращаемое значение

Возвращает массив разностей между соседними элементами.

### Пример

Запрос:

```
SELECT arrayDifference([1, 2, 3, 4]);
```

Результат:

```
arrayDifference([1, 2, 3, 4])  
[0,1,1,1]
```

Пример переполнения из-за результирующего типа Int64:

Запрос:

```
SELECT arrayDifference([0, 10000000000000000000]);
```

Результат:

```
arrayDifference([0, 10000000000000000000])  
[0,-8446744073709551616]
```

## arrayDistinct

Принимает массив, возвращает массив, содержащий уникальные элементы.

### Синтаксис

```
arrayDistinct(array)
```

### Аргументы

- **array** – массив.

### Возвращаемое значение

Возвращает массив, содержащий только уникальные элементы исходного массива.

### Пример

Запрос:

```
SELECT arrayDistinct([1, 2, 2, 3, 1]);
```

Ответ:

```
arrayDistinct([1, 2, 2, 3, 1])  
[1,2,3]
```

## arrayEnumerateDense(arr)

Возвращает массив того же размера, что и исходный массив, с индексами исходного массива, указывающими, где каждый элемент впервые появляется в исходном массиве.

Пример:

```
SELECT arrayEnumerateDense([10, 20, 10, 30])
```

```
arrayEnumerateDense([10, 20, 10, 30])  
[1,2,1,3]
```

## arrayIntersect(arr)

Принимает несколько массивов, возвращает массив с элементами, присутствующими во всех исходных массивах.

Пример:

```
SELECT  
    arrayIntersect([1, 2], [1, 3], [2, 3]) AS no_intersect,  
    arrayIntersect([1, 2], [1, 3], [1, 4]) AS intersect
```

```
no_intersect intersect
```

```
[] | [1] |
```

## arrayReduce

Применяет агрегатную функцию к элементам массива и возвращает ее результат. Имя агрегирующей функции передается как строка в одинарных кавычках 'max', 'sum'. При использовании параметрических агрегатных функций, параметр указывается после имени функции в круглых скобках 'uniqUpTo(6)'.

### Синтаксис

```
arrayReduce(agg_func, arr1, arr2, ..., arrN)
```

### Аргументы

- `agg_func` — Имя агрегатной функции, которая должна быть константой **string**.
- `arr` — Любое количество столбцов типа **array** в качестве параметров агрегатной функции.

### Возвращаемое значение

#### Пример

Запрос:

```
SELECT arrayReduce('max', [1, 2, 3]);
```

Результат:

```
arrayReduce('max', [1, 2, 3])  
3
```

Если агрегатная функция имеет несколько аргументов, то эту функцию можно применять к нескольким массивам одинакового размера.

#### Пример

Запрос:

```
SELECT arrayReduce('maxIf', [3, 5], [1, 0]);
```

Результат:

```
arrayReduce('maxIf', [3, 5], [1, 0])—  
3 |
```

Пример с параметрической агрегатной функцией:

Запрос:

```
SELECT arrayReduce('uniqUpTo(3)', [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]);
```

Результат:

```
arrayReduce('uniqUpTo(3)', [1, 2, 3, 4, 5, 6, 7, 8, 9, 10])—  
4 |
```

## arrayReduceInRanges

Применяет агрегатную функцию к элементам массива в заданных диапазонах и возвращает массив, содержащий результат, соответствующий каждому диапазону. Функция вернет тот же результат, что и несколько `arrayReduce(agg_func, arraySlice(arr1, index, length), ...)`.

### Синтаксис

```
arrayReduceInRanges(agg_func, ranges, arr1, arr2, ..., arrN)
```

### Аргументы

- `agg_func` — имя агрегатной функции, которая должна быть **строковой** константой.
- `ranges` — диапазоны для агрегирования, которые должны быть **массивом of кортежей** содержащих индекс и длину каждого диапазона.
- `arr` — любое количество столбцов типа **Array** в качестве параметров агрегатной функции.

### Возвращаемое значение

- Массив, содержащий результаты агрегатной функции для указанных диапазонов.

Тип: **Array**.

### Пример

Запрос:

```
SELECT arrayReduceInRanges(  
    'sum',  
    [(1, 5), (2, 3), (3, 4), (4, 4)],  
    [1000000, 200000, 30000, 4000, 500, 60, 7]  
) AS res
```

Результат:

```
res
[1234500,234000,34560,4567] |
```

## arrayReverse(arr)

Возвращает массив того же размера, что и исходный массив, содержащий элементы в обратном порядке.

Пример:

```
SELECT arrayReverse([1, 2, 3])
```

```
arrayReverse([1, 2, 3])—
[3,2,1]
```

## reverse(arr)

Синоним для «arrayReverse»

## arrayFlatten

Преобразует массив массивов в плоский массив.

Функция:

- Оперирует с массивами любой вложенности.
- Не изменяет массив, если он уже плоский.

Результирующий массив содержит все элементы исходных массивов.

### Синтаксис

```
flatten(array_of_arrays)
```

Синоним: flatten.

### Аргументы

- `array_of_arrays` — **массив** массивов. Например, `[[1,2,3], [4,5]]`.

### Примеры

```
SELECT flatten([[1], [2], [3]]);
```

```
flatten(array(array([1]), array([2], [3])))—
[1,2,3]
```

## arrayCompact

Удаляет последовательно повторяющиеся элементы из массива. Порядок результирующих значений определяется порядком в исходном массиве.

## Синтаксис

```
arrayCompact(arr)
```

## Аргументы

arr — **массив** для обхода.

## Возвращаемое значение

Массив без последовательных дубликатов.

Тип: `Array`.

## Пример

Запрос:

```
SELECT arrayCompact([1, 1, nan, nan, 2, 3, 3, 3]);
```

Результат:

```
arrayCompact([1, 1, nan, nan, 2, 3, 3, 3])—  
[1,nan,nan,2,3]
```

## arrayZip

Объединяет несколько массивов в один. Результирующий массив содержит соответственные элементы исходных массивов, сгруппированные в кортежи в указанном порядке аргументов.

## Синтаксис

```
arrayZip(arr1, arr2, ..., arrN)
```

## Аргументы

- arrN — **массив**.

Функция принимает любое количество массивов, которые могут быть различных типов. Все массивы должны иметь одинаковую длину.

## Возвращаемое значение

- Массив с элементами исходных массивов, сгруппированными в **кортежи**. Типы данных в кортежах соответствуют типам данных входных массивов и следуют в том же порядке, в котором переданы массивы.

Тип: **Массив**.

## Пример

Запрос:

```
SELECT arrayZip(['a', 'b', 'c'], [5, 2, 1]);
```

Результат:

```
arrayZip(['a', 'b', 'c'], [5, 2, 1])  
[('a',5),('b',2),('c',1)] |
```

## arrayMap(func, arr1, ...)

Возвращает массив, полученный на основе результатов применения функции `func` к каждому элементу массива `arr`.

Примеры:

```
SELECT arrayMap(x -> (x + 2), [1, 2, 3]) AS res;
```

```
res  
[3,4,5] |
```

Следующий пример показывает, как создать кортежи из элементов разных массивов:

```
SELECT arrayMap((x, y) -> (x, y), [1, 2, 3], [4, 5, 6]) AS res;
```

```
res  
[(1,4),(2,5),(3,6)] |
```

Функция `arrayMap` является [функцией высшего порядка](#) — в качестве первого аргумента ей нужно передать лямбда-функцию, и этот аргумент не может быть опущен.

## arrayFilter(func, arr1, ...)

Возвращает массив, содержащий только те элементы массива `arr1`, для которых функция `func` возвращает не 0.

Примеры:

```
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] |
```

Функция `arrayFilter` является **функцией высшего порядка** — в качестве первого аргумента ей нужно передать лямбда-функцию, и этот аргумент не может быть опущен.

## arrayFill(func, arr1, ...)

Перебирает `arr1` от первого элемента к последнему и заменяет `arr1[i]` на `arr1[i - 1]`, если `func` вернула 0. Первый элемент `arr1` остаётся неизменным.

Примеры:

```
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] |
```

Функция `arrayFill` является **функцией высшего порядка** — в качестве первого аргумента ей нужно передать лямбда-функцию, и этот аргумент не может быть опущен.

## arrayReverseFill(func, arr1, ...)

Перебирает `arr1` от последнего элемента к первому и заменяет `arr1[i]` на `arr1[i + 1]`, если `func` вернула 0. Последний элемент `arr1` остаётся неизменным.

Примеры:

```
SELECT arrayReverseFill(x -> not isNull(x), [1, null, 3, 11, 12, null, null, 5, 6, 14, null, null]) AS res
```

```
res  
[1,3,3,11,12,5,5,6,14,NULL,NULL] |
```

Функция `arrayReverseFill` является **функцией высшего порядка** — в качестве первого аргумента ей нужно передать лямбда-функцию, и этот аргумент не может быть опущен.

## arraySplit(func, arr1, ...)

Разделяет массив `arr1` на несколько. Если `func` возвращает не 0, то массив разделяется, а элемент помещается в левую часть. Массив не разбивается по первому элементу.

Примеры:

```
SELECT arraySplit((x, y) -> y, [1, 2, 3, 4, 5], [1, 0, 0, 1, 0]) AS res
```

```
res  
[[1,2,3],[4,5]] |
```

Функция `arraySplit` является **функцией высшего порядка** — в качестве первого аргумента ей нужно передать лямбда-функцию, и этот аргумент не может быть опущен.

## arrayReverseSplit(func, arr1, ...)

Разделяет массив `arr1` на несколько. Если `func` возвращает не 0, то массив разделяется, а элемент помещается в правую часть. Массив не разбивается по последнему элементу.

Примеры:

```
SELECT arrayReverseSplit((x, y) -> y, [1, 2, 3, 4, 5], [1, 0, 0, 1, 0]) AS res
```

```
res  
[[1],[2,3,4],[5]] |
```

Функция `arrayReverseSplit` является [функцией высшего порядка](#) — в качестве первого аргумента ей нужно передать лямбда-функцию, и этот аргумент не может быть опущен.

## arrayExists([func,] arr1, ...)

Возвращает 1, если существует хотя бы один элемент массива `arr`, для которого функция `func` возвращает не 0. Иначе возвращает 0.

Функция `arrayExists` является [функцией высшего порядка](#) — в качестве первого аргумента ей можно передать лямбда-функцию.

## arrayAll([func,] arr1, ...)

Возвращает 1, если для всех элементов массива `arr`, функция `func` возвращает не 0. Иначе возвращает 0.

Функция `arrayAll` является [функцией высшего порядка](#) — в качестве первого аргумента ей можно передать лямбда-функцию.

## arrayFirst(func, arr1, ...)

Возвращает первый элемент массива `arr1`, для которого функция `func` возвращает не 0.

Функция `arrayFirst` является [функцией высшего порядка](#) — в качестве первого аргумента ей нужно передать лямбда-функцию, и этот аргумент не может быть опущен.

## arrayFirstIndex(func, arr1, ...)

Возвращает индекс первого элемента массива `arr1`, для которого функция `func` возвращает не 0.

Функция `arrayFirstIndex` является [функцией высшего порядка](#) — в качестве первого аргумента ей нужно передать лямбда-функцию, и этот аргумент не может быть опущен.

## arrayMin

Возвращает значение минимального элемента в исходном массиве.

Если передана функция `func`, возвращается минимум из элементов массива, преобразованных этой функцией.

Функция `arrayMin` является [функцией высшего порядка](#) — в качестве первого аргумента ей можно передать лямбда-функцию.

### Синтаксис

```
arrayMin([func,] arr)
```

### Аргументы

- `func` — функция. [Expression](#).

- `arr` — массив. [Array](#).

## Возвращаемое значение

- Минимальное значение функции (или минимальный элемент массива).

Тип: если передана `func`, соответствует типу ее возвращаемого значения, иначе соответствует типу элементов массива.

## Примеры

Запрос:

```
SELECT arrayMin([1, 2, 4]) AS res;
```

Результат:

```
res  
1 |
```

Запрос:

```
SELECT arrayMin(x -> (-x), [1, 2, 4]) AS res;
```

Результат:

```
res  
-4 |
```

## arrayMax

Возвращает значение максимального элемента в исходном массиве.

Если передана функция `func`, возвращается максимум из элементов массива, преобразованных этой функцией.

Функция `arrayMax` является [функцией высшего порядка](#) — в качестве первого аргумента ей можно передать лямбда-функцию.

## Синтаксис

```
arrayMax([func,] arr)
```

## Аргументы

- `func` — функция. [Expression](#).

- `arr` — массив. [Array](#).

## Возвращаемое значение

- Максимальное значение функции (или максимальный элемент массива).

Тип: если передана `func`, соответствует типу ее возвращаемого значения, иначе соответствует типу элементов массива.

## Примеры

Запрос:

```
SELECT arrayMax([1, 2, 4]) AS res;
```

Результат:

```
res  
4 |
```

Запрос:

```
SELECT arrayMax(x -> (-x), [1, 2, 4]) AS res;
```

Результат:

```
res  
-1 |
```

## arraySum

Возвращает сумму элементов в исходном массиве.

Если передана функция `func`, возвращается сумма элементов массива, преобразованных этой функцией.

Функция `arraySum` является [функцией высшего порядка](#) — в качестве первого аргумента ей можно передать лямбда-функцию.

## Синтаксис

```
arraySum([func,] arr)
```

## Аргументы

- `func` — функция. [Expression](#).
- `arr` — массив. [Array](#).

## Возвращаемое значение

- Сумма значений функции (или сумма элементов массива).

Тип: для Decimal чисел в исходном массиве (если функция `func` была передана, то для чисел, преобразованных ею) — [Decimal128](#), для чисел с плавающей точкой — [Float64](#), для беззнаковых целых чисел — [UInt64](#), для целых чисел со знаком — [Int64](#).

## Примеры

Запрос:

```
SELECT arraySum([2, 3]) AS res;
```

Результат:

```
res  
5 |
```

Запрос:

```
SELECT arraySum(x -> x*x, [2, 3]) AS res;
```

Результат:

```
res  
13 |
```

## arrayAvg

Возвращает среднее значение элементов в исходном массиве.

Если передана функция `func`, возвращается среднее значение элементов массива, преобразованных этой функцией.

Функция `arrayAvg` является [функцией высшего порядка](#) — в качестве первого аргумента ей можно передать лямбда-функцию.

### Синтаксис

```
arrayAvg([func,] arr)
```

### Аргументы

- `func` — функция. [Expression](#).
- `arr` — массив. [Array](#).

### Возвращаемое значение

- Среднее значение функции (или среднее значение элементов массива).

Тип: [Float64](#).

### Примеры

Запрос:

```
SELECT arrayAvg([1, 2, 4]) AS res;
```

Результат:

```
res  
2.3333333333333335 |
```

Запрос:

```
SELECT arrayAvg(x -> (x * x), [2, 4]) AS res;
```

Результат:

```
res  
10 |
```

## Синтаксис

```
arraySum(arr)
```

## Возвращаемое значение

- Число.

Тип: **Int** или **Float**.

## Аргументы

- arr** — массив.

## Примеры

Запрос:

```
SELECT arraySum([2,3]) AS res;
```

Результат:

```
res  
5 |
```

Запрос:

```
SELECT arraySum(x -> x*x, [2, 3]) AS res;
```

Результат:

```
res  
13 |
```

## arrayCumSum([func,] arr1, ...)

Возвращает массив из частичных сумм элементов исходного массива (сумма с накоплением). Если указана функция **func**, то значения элементов массива преобразуются этой функцией перед суммированием.

Функция **arrayCumSum** является **функцией высшего порядка** - в качестве первого аргумента ей можно передать лямбда-функцию.

Пример:

```
SELECT arrayCumSum([1, 1, 1, 1]) AS res
```

```
res  
[1, 2, 3, 4] |
```

## arrayAUC

Вычисляет площадь под кривой.

### Синтаксис

```
arrayAUC(arr_scores, arr_labels)
```

### Аргументы

- `arr_scores` — оценка, которую дает модель предсказания.
- `arr_labels` — ярлыки выборок, обычно 1 для содержательных выборок и 0 для бессодержательных выборок.

### Возвращаемое значение

Значение площади под кривой.

Тип данных: `Float64`.

### Пример

Запрос:

```
SELECT arrayAUC([0.1, 0.4, 0.35, 0.8], [0, 0, 1, 1]);
```

Результат:

```
arrayAUC([0.1, 0.4, 0.35, 0.8], [0, 0, 1, 1])—  
0.75 |
```

## arrayProduct

Возвращает произведение элементов **массива**.

### Синтаксис

```
arrayProduct(arr)
```

### Аргументы

- `arr` — **массив** числовых значений.

### Возвращаемое значение

- Произведение элементов массива.

Тип: **Float64**.

## Примеры

Запрос:

```
SELECT arrayProduct([1,2,3,4,5,6]) as res;
```

Результат:

```
res  
720 |
```

Запрос:

```
SELECT arrayProduct([toDecimal64(1,8), toDecimal64(2,8), toDecimal64(3,8)]) as res, toTypeName(res);
```

Возвращаемое значение всегда имеет тип **Float64**. Результат:

```
res toTypeName(arrayProduct(array(toDecimal64(1, 8), toDecimal64(2, 8), toDecimal64(3, 8))))  
6 | Float64 |
```

## ФУНКЦИИ СРАВНЕНИЯ

Функции сравнения возвращают всегда 0 или 1 (UInt8).

Сравнивать можно следующие типы:

- числа;
- строки и фиксированные строки;
- даты;
- даты-с-временем;

внутри каждой группы, но не из разных групп.

Например, вы не можете сравнить дату со строкой. Надо использовать функцию преобразования строки в дату или наоборот.

Строки сравниваются побайтово. Более короткая строка меньше всех строк, начинающихся с неё и содержащих ещё хотя бы один символ.

Замечание. До версии 1.1.54134 сравнение знаковых и беззнаковых целых чисел производилось также, как в C++. То есть, вы могли получить неверный результат в таких случаях: `SELECT 9223372036854775807 > -1`. С версии 1.1.54134 поведение изменилось и стало математически корректным.

`equals`, оператор `a = b` и `a == b`

`notEquals`, оператор `a != b` и `a <> b`

`less`, оператор `<`

`greater`, оператор `>`

`lessOrEquals`, оператор `<=`

`greaterOrEquals`, оператор `>=`

## Логические функции

Логические функции производят логические операции над любыми числовыми типами, а возвращают число типа `UInt8`, равное 0, 1, а в некоторых случаях `NULL`.

Ноль в качестве аргумента считается `ложью`, а любое ненулевое значение — `истиной`.

### `and`

Вычисляет результат логической конъюнкции между двумя и более значениями. Соответствует [оператору логического "И"](#).

#### Синтаксис

```
and(val1, val2...)
```

Чтобы вычислять функцию `and` по короткой схеме, используйте настройку [short\\_circuit\\_function\\_evaluation](#). Если настройка включена, то выражение `vali` вычисляется только для строк, где условие `(val1 AND val2 AND ... AND val{i-1})` верно. Например, при выполнении запроса `SELECT and(number = 2, intDiv(1, number)) FROM numbers(10)` не будет сгенерировано исключение из-за деления на ноль.

#### Аргументы

- `val1, val2, ...` — список из как минимум двух значений. [Int](#), [UInt](#), [Float](#) или [Nullable](#).

#### Возвращаемое значение

- `0`, если среди аргументов есть хотя бы один нуль.
- `NULL`, если среди аргументов нет нулей, но есть хотя бы один `NULL`.
- `1`, в остальных случаях.

Тип: [UInt8](#) или [Nullable\(UInt8\)](#).

#### Пример

Запрос:

```
SELECT and(0, 1, -2);
```

Результат:

```
and(0, 1, -2)
  0 |
```

Со значениями `NULL`:

```
SELECT and(NULL, 1, 10, -2);
```

Результат:

```
and(NULL, 1, 10, -2)  
      |  
      |
```

Or

Вычисляет результат логической дизъюнкции между двумя и более значениями. Соответствует оператору логического "ИЛИ".

## Синтаксис

```
and(val1, val2...)
```

Чтобы вычислять функцию `or` по короткой схеме, используйте настройку `short_circuit_function_evaluation`. Если настройка включена, то выражение `vali` вычисляется только для строк, где условие `((NOT val1) AND (NOT val2) AND ... AND (NOT val{i-1}))` верно. Например, при выполнении запроса `SELECT or(number = 0, intDiv(1, number) != 0) FROM numbers(10)` не будет сгенерировано исключение из-за деления на ноль.

## Аргументы

- `val1, val2, ...` — список из как минимум двух значений. `Int`, `UInt`, `Float` или `Nullable`.

## Returned value

- `1`, если среди аргументов есть хотя бы одно ненулевое число.
- `0`, если среди аргументов только нули.
- `NULL`, если среди аргументов нет ненулевых значений, и есть `NULL`.

Тип: `UInt8` или `Nullable(UInt8)`.

## Пример

Запрос:

```
SELECT or(1, 0, 0, 2, NULL);
```

Результат:

```
or(1, 0, 0, 2, NULL)  
      |  
      |
```

Со значениями `NULL`:

```
SELECT or(0, NULL);
```

Результат:

```
└─or(0, NULL)─  
   ┌─NULL─┐
```

## not

Вычисляет результат логического отрицания аргумента. Соответствует [оператору логического отрицания](#).

### Синтаксис

```
not(val);
```

### Аргументы

- `val` — значение [Int](#), [UInt](#), [Float](#) или [Nullable](#).

### Возвращаемое значение

- 1, если `val` — это 0.
- 0, если `val` — это ненулевое число.
- NULL, если `val` — это NULL.

Тип: [UInt8](#) или [Nullable\(UInt8\)](#).

### Пример

Запрос:

```
SELECT NOT(1);
```

Результат:

```
└─not(1)─  
   ┌─0─┐
```

## xor

Вычисляет результат логической исключающей дизъюнкции между двумя и более значениями. При более чем двух значениях функция работает так: сначала вычисляет XOR для первых двух значений, а потом использует полученный результат при вычислении XOR со следующим значением и так далее.

### Синтаксис

```
xor(val1, val2...)
```

### Аргументы

- `val1, val2, ...` — список из как минимум двух значений [Int](#), [UInt](#), [Float](#) или [Nullable](#).

### Returned value

- 1, для двух значений: если одно из значений является нулем, а второе нет.

- 0, для двух значений: если оба значения одновременно нули или ненулевые числа.
- NULL, если среди аргументов хотя бы один NULL.

Тип: `UInt8` or `Nullable(UInt8)`.

## Пример

Запрос:

```
SELECT xor(0, 1, 1);
```

Результат:

```
+-----+  
| xor(0, 1, 1) |  
+-----+  
| 0 |  
+-----+
```

# ФУНКЦИИ преобразования типов

## Общие проблемы преобразования чисел

При преобразовании значения из одного типа в другой необходимо помнить, что в общем случае это небезопасная операция, которая может привести к потере данных. Потеря данных может произойти при попытке сконвертировать тип данных значения от большего к меньшему или при конвертировании между различными классами типов данных.

Поведение ClickHouse при конвертировании похоже на [поведение C++ программ](#).

## `toInt(8|16|32|64|128|256)`

Преобразует входное значение к типу `Int`. Семейство функций включает:

- `toInt8(expr)` — возвращает значение типа `Int8`.
- `toInt16(expr)` — возвращает значение типа `Int16`.
- `toInt32(expr)` — возвращает значение типа `Int32`.
- `toInt64(expr)` — возвращает значение типа `Int64`.
- `toInt128(expr)` — возвращает значение типа `Int128`.
- `toInt256(expr)` — возвращает значение типа `Int256`.

## Аргументы

- `expr` — [выражение](#) возвращающее число или строку с десятичным представлением числа. Бинарное, восьмеричное и шестнадцатеричное представление числа не поддерживаются. Ведущие нули обрезаются.

## Возвращаемое значение

Целое число типа `Int8`, `Int16`, `Int32`, `Int64`, `Int128` или `Int256`.

Функции используют [округление к нулю](#), т.е. обрезают дробную часть числа.

Поведение функций для аргументов `Nan` и `Inf` не определено. При использовании функций помните о возможных проблемах при [преобразовании чисел](#).

## **Пример**

Запрос:

```
SELECT tolnt64(nan), tolnt32(32), tolnt16('16'), tolnt8(8.8);
```

Результат:

tolnt64(nan)	tolnt32(32)	tolnt16('16')	tolnt8(8.8)
-9223372036854775808	32	16	8

## **tolnt(8|16|32|64|128|256)OrZero**

Принимает аргумент типа String и пытается его распарсить в Int(8|16|32|64|128|256). Если не удалось - возвращает 0.

## **Пример**

Запрос:

```
SELECT tolnt64OrZero('123123'), tolnt8OrZero('123qwe123');
```

Результат:

tolnt64OrZero('123123')	tolnt8OrZero('123qwe123')
123123	0

## **tolnt(8|16|32|64|128|256)OrNull**

Принимает аргумент типа String и пытается его распарсить в Int(8|16|32|64|128|256). Если не удалось - возвращает NULL.

## **Пример**

Запрос:

```
SELECT tolnt64OrNull('123123'), tolnt8OrNull('123qwe123');
```

Результат:

tolnt64OrNull('123123')	tolnt8OrNull('123qwe123')
123123	NULL

## **tolnt(8|16|32|64|128|256)OrDefault**

Принимает аргумент типа String и пытается его распарсить в Int(8|16|32|64|128|256). Если не удалось — возвращает значение по умолчанию.

## **Пример**

Запрос:

```
SELECT toInt64OrDefault('123123', cast('-1' as Int64)), toInt8OrDefault('123qwe123', cast('-1' as Int8));
```

Результат:

toInt64OrDefault('123123', CAST('-1', 'Int64'))	toInt8OrDefault('123qwe123', CAST('-1', 'Int8'))
123123	-1

## toUInt(8|16|32|64|256)

Преобразует входное значение к типу [UInt](#). Семейство функций включает:

- `toUInt8(expr)` — возвращает значение типа `UInt8`.
- `toUInt16(expr)` — возвращает значение типа `UInt16`.
- `toUInt32(expr)` — возвращает значение типа `UInt32`.
- `toUInt64(expr)` — возвращает значение типа `UInt64`.
- `toUInt256(expr)` — возвращает значение типа `UInt256`.

### Аргументы

- `expr` — [выражение](#) возвращающее число или строку с десятичным представлением числа. Бинарное, восьмеричное и шестнадцатеричное представление числа не поддерживаются. Ведущие нули обрезаются.

### Возвращаемое значение

Целое число типа `UInt8`, `UInt16`, `UInt32`, `UInt64` или `UInt256`.

Функции используют [округление к нулю](#), т.е. обрезают дробную часть числа.

Поведение функций для аргументов [NaN](#) и [Inf](#) не определено. Если передать строку, содержащую отрицательное число, например `'-32'`, ClickHouse генерирует исключение. При использовании функций помните о возможных проблемах при [преобразовании чисел](#).

### Пример

Запрос:

```
SELECT toUInt64(nan), toUInt32(-32), toUInt16('16'), toUInt8(8.8);
```

Результат:

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`

Синоним: `DATE`.

`toDateOrZero`

`toDateOrNull`

`toDateOrDefault`

`toDateTime`

`toDateTimeOrZero`

`toDateTimeOrNull`

`toDateTimeOrDefault`

`toDate32`

Конвертирует аргумент в значение типа `Date32`. Если значение выходит за границы диапазона, возвращается пограничное значение `Date32`. Если аргумент имеет тип `Date`, учитываются границы типа `Date`.

## Синтаксис

```
toDate32(value)
```

## Аргументы

- `value` — Значение даты. `String`, `UInt32` или `Date`.

## Возвращаемое значение

- Календарная дата.

Тип: `Date32`.

## Пример

1. Значение находится в границах диапазона:

```
SELECT toDate32('1955-01-01') AS value, toTypeName(value);
```

value	→	toTypeName(toDate32('1925-01-01'))	→
1955-01-01		Date32	

2. Значение выходит за границы диапазона:

```
SELECT toDate32('1924-01-01') AS value, typeName(value);
```

value	typeName(toDate32('1925-01-01'))
1925-01-01	Date32

3. С аргументом типа Date:

```
SELECT toDate32(toDate('1924-01-01')) AS value, typeName(value);
```

value	typeName(toDate32(toDate('1924-01-01')))
1970-01-01	Date32

## toDate32OrZero

То же самое, что и [toDate32](#), но возвращает минимальное значение типа **Date32**, если получен недопустимый аргумент.

### Пример

Запрос:

```
SELECT toDate32OrZero('1924-01-01'), toDate32OrZero('');
```

Результат:

toDate32OrZero('1924-01-01')	toDate32OrZero('')
1925-01-01	1925-01-01

## toDate32OrNull

То же самое, что и [toDate32](#), но возвращает **NULL**, если получен недопустимый аргумент.

### Пример

Запрос:

```
SELECT toDate32OrNull('1955-01-01'), toDate32OrNull('');
```

Результат:

toDate32OrNull('1955-01-01')	toDate32OrNull('')
1955-01-01	NULL

## toDate32OrDefault

Конвертирует аргумент в значение типа **Date32**. Если значение выходит за границы диапазона, возвращается нижнее пограничное значение **Date32**. Если аргумент имеет тип **Date**, учитываются границы типа **Date**. Возвращает значение по умолчанию, если получен недопустимый аргумент.

## Пример

Запрос:

```
SELECT
    toDate32OrDefault('1930-01-01', toDate32('2020-01-01')),
    toDate32OrDefault('xx1930-01-01', toDate32('2020-01-01'));
```

Результат:

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)

Преобразует **value** к типу данных **Decimal** с точностью **S**. **value** может быть числом или строкой. Параметр **S** (scale) задаёт число десятичных знаков.

- `toDecimal32(value, S)`
- `toDecimal64(value, S)`
- `toDecimal128(value, S)`
- `toDecimal256(value, S)`

## toDecimal(32|64|128|256)OrNull

Преобразует входную строку в значение с типом данных **Nullable (Decimal (P, S))**. Семейство функций включает в себя:

- `toDecimal32OrNull(expr, S)` — Возвращает значение типа `Nullable(Decimal32(S))`.
- `toDecimal64OrNull(expr, S)` — Возвращает значение типа `Nullable(Decimal64(S))`.
- `toDecimal128OrNull(expr, S)` — Возвращает значение типа `Nullable(Decimal128(S))`.
- `toDecimal256OrNull(expr, S)` — Возвращает значение типа `Nullable(Decimal256(S))`.

Эти функции следует использовать вместо функций `toDecimal*()`, если при ошибке обработки входного значения вы хотите получать `NULL` вместо исключения.

## Аргументы

- **expr** — **выражение**, возвращающее значение типа **String**. ClickHouse ожидает текстовое представление десятичного числа. Например, `'1.111'`.
- **S** — количество десятичных знаков в результирующем значении.

## Возвращаемое значение

Значение типа **Nullable(Decimal(P,S))**. Значение содержит:

- Число с S десятичными знаками, если ClickHouse распознал число во входной строке.
- NULL, если ClickHouse не смог распознать число во входной строке или входное число содержит больше чем S десятичных знаков.

## Примеры

Запрос:

```
SELECT toDecimal32OrNull(toString(-1.111), 5) AS val, typeName(val);
```

Результат:

val	typeName(toDecimal32OrNull(toString(-1.111), 5))
-1.111100	Nullable(Decimal(9, 5))

Запрос:

```
SELECT toDecimal32OrNull(toString(-1.111), 2) AS val, typeName(val);
```

Результат:

val	typeName(toDecimal32OrNull(toString(-1.111), 2))
NULL	Nullable(Decimal(9, 2))

## toDecimal(32|64|128|256)OrDefault

Преобразует входную строку в значение с типом данных **Decimal(P,S)**. Семейство функций включает в себя:

- `toDecimal32OrDefault(expr, S)` — возвращает значение типа `Decimal32(S)`.
- `toDecimal64OrDefault(expr, S)` — возвращает значение типа `Decimal64(S)`.
- `toDecimal128OrDefault(expr, S)` — возвращает значение типа `Decimal128(S)`.
- `toDecimal256OrDefault(expr, S)` — возвращает значение типа `Decimal256(S)`.

Эти функции следует использовать вместо функций `toDecimal*`(), если при ошибке обработки входного значения вы хотите получать значение по умолчанию вместо исключения.

## Аргументы

- `expr` — выражение, возвращающее значение типа `String`. ClickHouse ожидает текстовое представление десятичного числа. Например, '`1.111`'.
- `S` — количество десятичных знаков в результирующем значении.

## Возвращаемое значение

Значение типа `Decimal(P,S)`. Значение содержит:

- Число с S десятичными знаками, если ClickHouse распознал число во входной строке.
- Значение по умолчанию типа `Decimal(P,S)`, если ClickHouse не смог распознать число во входной строке или входное число содержит больше чем S десятичных знаков.

## Примеры

Запрос:

```
SELECT toDecimal32OrDefault(toString(-1.111), 5) AS val, toTypeName(val);
```

Результат:

val	toTypeName(toDecimal32OrDefault(toString(-1.111), 5))
-1.111	Decimal(9, 5)

Запрос:

```
SELECT toDecimal32OrDefault(toString(-1.111), 2) AS val, toTypeName(val);
```

Результат:

val	toTypeName(toDecimal32OrDefault(toString(-1.111), 2))
0	Decimal(9, 2)

## toDecimal(32|64|128|256)OrZero

Преобразует тип входного значения в **Decimal (P, S)**. Семейство функций включает в себя:

- `toDecimal32OrZero( expr, S )` — возвращает значение типа `Decimal32(S)`.
- `toDecimal64OrZero( expr, S )` — возвращает значение типа `Decimal64(S)`.
- `toDecimal128OrZero( expr, S )` — возвращает значение типа `Decimal128(S)`.
- `toDecimal256OrZero( expr, S )` — возвращает значение типа `Decimal256(S)`.

Эти функции следует использовать вместо функций `toDecimal*`(), если при ошибке обработки входного значения вы хотите получать 0 вместо исключения.

### Аргументы

- `expr` — **выражение**, возвращающее значение типа `String`. ClickHouse ожидает текстовое представление десятичного числа. Например, `'1.111'`.
- `S` — количество десятичных знаков в результирующем значении.

### Возвращаемое значение

Значение типа `Nullable(Decimal(P,S))`. `P` равно числовой части имени функции. Например, для функции `toDecimal32OrZero`, `P = 32`. Значение содержит:

- Число с `S` десятичными знаками, если ClickHouse распознал число во входной строке.
- 0 с `S` десятичными знаками, если ClickHouse не смог распознать число во входной строке или входное число содержит больше чем `S` десятичных знаков.

## Пример

Запрос:

```
SELECT toDecimal32OrZero(toString(-1.111), 5) AS val, toTypeName(val);
```

Результат:

val	toTypeName(toDecimal32OrZero(toString(-1.111), 5))
-1.11100	Decimal(9, 5)

Запрос:

```
SELECT toDecimal32OrZero(toString(-1.111), 2) AS val, toTypeName(val);
```

Результат:

val	toTypeName(toDecimal32OrZero(toString(-1.111), 2))
0.00	Decimal(9, 2)

## toString

Функции преобразования между числами, строками (но не фиксированными строками), датами и датами-с-временем.

Все эти функции принимают один аргумент.

При преобразовании в строку или из строки, производится форматирование или парсинг значения по тем же правилам, что и для формата TabSeparated (и почти всех остальных текстовых форматов). Если распарсить строку не удаётся - кидается исключение и выполнение запроса прерывается.

При преобразовании даты в число или наоборот, дате соответствует число дней от начала unix эпохи.

При преобразовании даты-с-временем в число или наоборот, дате-с-временем соответствует число секунд от начала unix эпохи.

Форматы даты и даты-с-временем для функций toDate/toDateTime определены следующим образом:

YYYY-MM-DD
YYYY-MM-DD hh:mm:ss

В качестве исключения, если делается преобразование из числа типа UInt32, Int32, UInt64, Int64 в Date, и если число больше или равно 65536, то число рассматривается как unix timestamp (а не как число дней) и округляется до даты. Это позволяет поддержать распространённый случай, когда пишут toDate(unix\_timestamp), что иначе было бы ошибкой и требовало бы написания более громоздкого toDate(toDateTime(unix\_timestamp))

Преобразование между датой и датой-с-временем производится естественным образом: добавлением нулевого времени или отбрасыванием времени.

Преобразование между числовыми типами производится по тем же правилам, что и присваивание между разными числовыми типами в C++.

Дополнительно, функция `toString` от аргумента типа `DateTime` может принимать второй аргумент `String` - имя тайм-зоны. Пример: `Asia/Yekaterinburg` В этом случае, форматирование времени производится согласно указанной тайм-зоне.

## Пример

Запрос:

```
SELECT
    now() AS now_local,
    toString(now(), 'Asia/Yekaterinburg') AS now_yekat;
```

Результат:

now_local	now_yekat
2016-06-15 00:11:21	2016-06-15 02:11:21

Также смотрите функцию `toUnixTimestamp`.

## toFixedString(s, N)

Преобразует аргумент типа String в тип FixedString(N) (строку фиксированной длины N). N должно быть константой.

Если строка имеет меньше байт, чем N, то она дополняется нулевыми байтами справа. Если строка имеет больше байт, чем N - кидается исключение.

## toStringCutToZero(s)

Принимает аргумент типа String или FixedString. Возвращает String, вырезая содержимое строки до первого найденного нулевого байта.

## Примеры

Запрос:

```
SELECT toFixedString('foo', 8) AS s, toStringCutToZero(s) AS s_cut;
```

Результат:

s	s_cut
foo\0\0\0\0\0   foo	\0

Запрос:

```
SELECT toFixedString('foo\0bar', 8) AS s, toStringCutToZero(s) AS s_cut;
```

Результат:

s	s_cut
foo\0bar\0   foo	\0

## reinterpretAsUInt(8|16|32|64)

## reinterpretAsInt(8|16|32|64)

## reinterpretAsFloat(32|64)

## reinterpretAsDate

## reinterpretAsDateTime

Функции принимают строку и интерпретируют байты, расположенные в начале строки, как число в host order (little endian). Если строка имеет недостаточную длину, то функции работают так, как будто строка дополнена необходимым количеством нулевых байт. Если строка длиннее, чем нужно, то лишние байты игнорируются. Дата интерпретируется, как число дней с начала unix-эпохи, а дата-с-временем - как число секунд с начала unix-эпохи.

## reinterpretAsString

Функция принимает число или дату или дату-с-временем и возвращает строку, содержащую байты, представляющие соответствующее значение в host order (little endian). При этом, отбрасываются нулевые байты с конца. Например, значение 255 типа UInt32 будет строкой длины 1 байт.

## reinterpretAsUUID

Функция принимает строку из 16 байт и интерпретирует ее байты в порядок от старшего к младшему. Если строка имеет недостаточную длину, то функция работает так, как будто строка дополнена необходимым количеством нулевых байтов с конца. Если строка длиннее, чем 16 байтов, то лишние байты с конца игнорируются.

### Синтаксис

```
reinterpretAsUUID(fixed_string)
```

### Аргументы

- fixed\_string — строка с big-endian порядком байтов. [FixedString](#).

### Возвращаемое значение

- Значение типа [UUID](#).

### Примеры

Интерпретация строки как UUID.

Запрос:

```
SELECT reinterpretAsUUID(reverse(unhex('000102030405060708090a0b0c0d0e0f')));
```

Результат:

```
└─ reinterpretAsUUID(reverse(unhex('000102030405060708090a0b0c0d0e0f'))) ─  
    08090a0b-0c0d-0e0f-0001-020304050607 |
```

Переход в UUID и обратно.

Запрос:

```
WITH
    generateUUIDv4() AS uuid,
    identity(lower(hex(reverse(reinterpretAsString(uuid))))) AS str,
    reinterpretAsUUID(reverse(unhex(str))) AS uuid2
SELECT uuid = uuid2;
```

Результат:

```
equals(uuid, uuid2)─
  1 |
```

## reinterpret(x, T)

Использует ту же самую исходную последовательность байтов в памяти для значения `x` и интерпретирует ее как конечный тип данных `T`.

### Синтаксис

```
reinterpret(x, type)
```

### Аргументы

- `x` — любой тип данных.
- `type` — конечный тип данных. [String](#).

### Возвращаемое значение

- Значение конечного типа данных.

### Примеры

Запрос:

```
SELECT reinterpret(toInt8(-1), 'UInt8') as int_to_uint,
       reinterpret(toInt8(1), 'Float32') as int_to_float,
       reinterpret('1', 'UInt32') as string_to_int;
```

Результат:

```
int_to_uint─int_to_float─string_to_int─
  255 |      1e-45 |        49 |
```

## CAST(x, T)

Преобразует входное значение к указанному типу данных. В отличие от функции `reinterpret` `CAST` пытается представить то же самое значение в новом типе данных. Если преобразование невозможно, то возникает исключение.

Поддерживается несколько вариантов синтаксиса.

### Синтаксис

```
CAST(x, T)
CAST(x AS t)
x::t
```

## Аргументы

- `x` — значение, которое нужно преобразовать. Может быть любого типа.
- `T` — имя типа данных. [String](#).
- `t` — тип данных.

## Возвращаемое значение

- Преобразованное значение.

## Примечание

Если входное значение выходит за границы нового типа, то результат переполняется.  
Например, `CAST(-1, 'UInt8')` возвращает 255.

## Примеры

Запрос:

```
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;
```

Результат:

cast_int_to_uint	cast_float_to_decimal	cast_string_to_int
255	1.50	1

Запрос:

```
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;
```

Результат:

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

Преобразование в `FixedString(N)` работает только для аргументов типа [String](#) или [FixedString](#).

Поддерживается преобразование к типу [Nullable](#) и обратно.

## Примеры

Запрос:

```
SELECT toTypeName(x) FROM t_null;
```

Результат:

```
toTypeName(x)
└─ Int8
  └─ Int8
```

Запрос:

```
SELECT toTypeName(CAST(x, 'Nullable(UInt16)')) FROM t_null;
```

Результат:

```
toTypeName(CAST(x, 'Nullable(UInt16)'))
└─ Nullable(UInt16)
  └─ Nullable(UInt16)
```

## Смотрите также

- Настройка [cast\\_keep\\_nullable](#)

## accurateCast(x, T)

Преобразует входное значение x в указанный тип данных T.

В отличие от функции [cast\(x, T\)](#), accurateCast не допускает переполнения при преобразовании числовых типов. Например, `accurateCast(-1, 'UInt8')` вызовет исключение.

## Примеры

Запрос:

```
SELECT cast(-1, 'UInt8') as uint8;
```

Результат:

```
uint8
└─ 255
```

Запрос:

```
SELECT accurateCast(-1, 'UInt8') as uint8;
```

Результат:

```
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)

Преобразует входное значение `x` в указанный тип данных `T`.

Всегда возвращает тип **Nullable**. Если исходное значение не может быть преобразовано к целевому типу, возвращает **NULL**.

## Синтаксис

```
accurateCastOrNull(x, T)
```

## Аргументы

- `x` — входное значение.
- `T` — имя возвращаемого типа данных.

## Возвращаемое значение

- Значение, преобразованное в указанный тип `T`.

## Примеры

Запрос:

```
SELECT toTypeName(accurateCastOrNull(5, 'UInt8'));
```

Результат:

```
toTypeName(accurateCastOrNull(5, 'UInt8'))  
 Nullable(UInt8)
```

Запрос:

```
SELECT  
    accurateCastOrNull(-1, 'UInt8') as uint8,  
    accurateCastOrNull(128, 'Int8') as int8,  
    accurateCastOrNull('Test', 'FixedString(2)') as fixed_string;
```

Результат:

```
uint8 int8 fixed_string  
NULL NULL NULL
```

## accurateCastOrDefault(x, T[, default\_value])

Преобразует входное значение `x` в указанный тип данных `T`. Если исходное значение не может быть преобразовано к целевому типу, возвращает значение по умолчанию или `default_value`, если оно указано.

## Синтаксис

```
accurateCastOrDefault(x, T)
```

## Аргументы

- `x` — входное значение.

- `T` — имя возвращаемого типа данных.
- `default_value` — значение по умолчанию возвращаемого типа данных.

## Возвращаемое значение

- Значение, преобразованное в указанный тип `T`.

## Пример

Запрос:

```
SELECT toTypeName(accurateCastOrDefault(5, 'UInt8'));
```

Результат:

```
toTypeName(accurateCastOrDefault(5, 'UInt8'))  
 UInt8
```

Запрос:

```
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;
```

Результат:

```
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)

Приводит аргумент из числового типа данных к типу данных [IntervalType](#).

## Синтаксис

```
toIntervalSecond(number)  
toIntervalMinute(number)  
toIntervalHour(number)  
toIntervalDay(number)  
toIntervalWeek(number)  
toIntervalMonth(number)  
toIntervalQuarter(number)  
toIntervalYear(number)
```

## Аргументы

- `number` — длительность интервала. Положительное целое число.

## Возвращаемые значения

- Значение с типом данных `Interval`.

## Пример

Запрос:

```
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;
```

Результат:

```
+-----+-----+
| plus(date, interval_week) | plus(date, interval_to_week) |
| 2019-01-08 | 2019-01-08 |
```

## parseDateTimeBestEffort

## parseDateTime32BestEffort

Преобразует дату и время в **строковом** представлении к типу данных **DateTime**.

Функция распознаёт форматы **ISO 8601**, **RFC 1123** - **5.2.14 RFC-822 Date and Time Specification**, формат даты времени ClickHouse's а также некоторые другие форматы.

### Синтаксис

```
parseDateTimeBestEffort(time_string[, time_zone])
```

### Аргументы

- `time_string` — строка, содержащая дату и время для преобразования. **String**.
- `time_zone` — часовой пояс. Функция анализирует `time_string` в соответствии с заданным часовым поясом. **String**.

### Поддерживаемые нестандартные форматы

- **Unix timestamp** в строковом представлении. 9 или 10 символов.
- Стока с датой и временем: `YYYYMMDDhhmmss`, `DD/MM/YYYY hh:mm:ss`, `DD-MM-YY hh:mm`, `YYYY-MM-DD hh:mm:ss`, etc.
- Стока с датой, но без времени: `YYYY`, `YYYYMM`, `YYYY*MM`, `DD/MM/YYYY`, `DD-MM-YY` и т.д.
- Стока с временем, и с днём: `DD`, `DD hh`, `DD hh:mm`. В этом случае `YYYY-MM` принимается равным `2000-01`.
- Стока, содержащая дату и время вместе с информацией о часовом поясе: `YYYY-MM-DD hh:mm:ss ±h:mm`, и т.д. Например, `2020-12-12 17:36:00 -5:00`.

Для всех форматов с разделителями функция распознаёт названия месяцев, выраженных в виде полного англоязычного имени месяца или в виде первых трёх символов имени месяца. Примеры: `24/DEC/18`, `24-Dec-18`, `01-September-2018`.

### Возвращаемое значение

- `time_string` преобразованная к типу данных **DateTime**.

### Примеры

Запрос:

```
SELECT parseDateTimeBestEffort('12/12/2020 12:12:57')
AS parseDateTimeBestEffort;
```

Результат:

```
parseDateTimeBestEffort—
2020-12-12 12:12:57 |
```

Запрос:

```
SELECT parseDateTimeBestEffort('Sat, 18 Aug 2018 07:22:16 GMT', 'Europe/Moscow')
AS parseDateTimeBestEffort;
```

Результат:

```
parseDateTimeBestEffort—
2018-08-18 10:22:16 |
```

Запрос:

```
SELECT parseDateTimeBestEffort('1284101485')
AS parseDateTimeBestEffort;
```

Результат:

```
parseDateTimeBestEffort—
2015-07-07 12:04:41 |
```

Запрос:

```
SELECT parseDateTimeBestEffort('2018-12-12 10:12:12')
AS parseDateTimeBestEffort;
```

Результат:

```
parseDateTimeBestEffort—
2018-12-12 10:12:12 |
```

Запрос:

```
SELECT parseDateTimeBestEffort('10 20:19');
```

Результат:

```
parseDateTimeBestEffort('10 20:19')—
2000-01-10 20:19:00 |
```

## Смотрите также

- Информация о формате ISO 8601 от @xkcd
- RFC 1123
- toDate
- toDateTime

## parseDateTimeBestEffortUS

Эта функция похожа на 'parseDateTimeBestEffort', но разница состоит в том, что в она предполагает американский формат даты (MM/DD/YYYY etc.) в случае неоднозначности.

### Синтаксис

```
parseDateTimeBestEffortUS(time_string [, time_zone])
```

### Аргументы

- `time_string` — строка, содержащая дату и время для преобразования. [String](#).
- `time_zone` — часовой пояс. Функция анализирует `time_string` в соответствии с часовым поясом. [String](#).

### Поддерживаемые нестандартные форматы

- Стока, содержащая 9-10 цифр [unix timestamp](#).
- Стока, содержащая дату и время: `YYYYMMDDhhmmss`, `MM/DD/YYYY hh:mm:ss`, `MM-DD-YY hh:mm`, `YYYY-MM-DD hh:mm:ss`, etc.
- Стока с датой, но без времени: `YYYY`, `YYYYMM`, `YYYY*MM`, `MM/DD/YYYY`, `MM-DD-YY` etc.
- Стока, содержащая день и время: `DD`, `DD hh`, `DD hh:mm`. В этом случае `YYYY-MM` заменяется на `2000-01`.
- Стока, содержащая дату и время, а также информацию о часовом поясе: `YYYY-MM-DD hh:mm:ss ±h:mm` и т.д. Например, `2020-12-12 17:36:00 -5:00`.

### Возвращаемое значение

- `time_string` преобразован в тип данных `DateTime`.

### Примеры

Запрос:

```
SELECT parseDateTimeBestEffortUS('09/12/2020 12:12:57')
AS parseDateTimeBestEffortUS;
```

Результат:

```
parseDateTimeBestEffortUS
2020-09-12 12:12:57 |
```

Запрос:

```
SELECT parseDateTimeBestEffortUS('09-12-2020 12:12:57')
AS parseDateTimeBestEffortUS;
```

Результат:

```
parseDateTimeBestEffortUS
2020-09-12 12:12:57 |
```

Запрос:

```
SELECT parseDateTimeBestEffortUS('09.12.2020 12:12:57')
AS parseDateTimeBestEffortUS;
```

Результат:

```
parseDateTimeBestEffortUS
2020-09-12 12:12:57 |
```

## parseDateTimeBestEffortOrNull

## parseDateTime32BestEffortOrNull

Работает также как [parseDateTimeBestEffort](#), но возвращает `NULL` когда получает формат даты который не может быть обработан.

## parseDateTimeBestEffortOrZero

## parseDateTime32BestEffortOrZero

Работает аналогично функции [parseDateTimeBestEffort](#), но возвращает нулевое значение, если формат даты не может быть обработан.

## parseDateTimeBestEffortUSOrNull

Работает аналогично функции [parseDateTimeBestEffortUS](#), но в отличие от нее возвращает `NULL`, если входная строка не может быть преобразована в тип данных [DateTime](#).

### Синтаксис

```
parseDateTimeBestEffortUSOrNull(time_string[, time_zone])
```

### Аргументы

- `time_string` — строка, содержащая дату или дату со временем для преобразования. Дата должна быть в американском формате (`MM/DD/YYYY` и т.д.). [String](#).
- `time_zone` — [часовой пояс](#). Функция анализирует `time_string` в соответствии с заданным часовым поясом. Опциональный параметр. [String](#).

### Поддерживаемые нестандартные форматы

- Стока в формате [unix timestamp](#), содержащая 9-10 цифр.

- Стока, содержащая дату и время: `YYYYMMDDhhmmss`, `MM/DD/YYYY hh:mm:ss`, `MM-DD-YY hh:mm`, `YYYY-MM-DD hh:mm:ss` и т.д.
- Стока, содержащая дату без времени: `YYYY`, `YYYYMM`, `YYYY*MM`, `MM/DD/YYYY`, `MM-DD-YY` и т.д.
- Стока, содержащая день и время: `DD`, `DD hh`, `DD hh:mm`. В этом случае `YYYY-MM` заменяется на `2000-01`.
- Стока, содержащая дату и время, а также информацию о часовом поясе: `YYYY-MM-DD hh:mm:ss ±h:mm` и т.д. Например, `2020-12-12 17:36:00 -5:00`.

## Возвращаемые значения

- `time_string`, преобразованная в тип данных `DateTime`.
- `NULL`, если входная строка не может быть преобразована в тип данных `DateTime`.

## Примеры

Запрос:

```
SELECT parseDateTimeBestEffortUSOrNull('02/10/2021 21:12:57') AS parseDateTimeBestEffortUSOrNull;
```

Результат:

```
parseDateTimeBestEffortUSOrNull—  
2021-02-10 21:12:57 |
```

Запрос:

```
SELECT parseDateTimeBestEffortUSOrNull('02-10-2021 21:12:57 GMT', 'Europe/Moscow') AS  
parseDateTimeBestEffortUSOrNull;
```

Результат:

```
parseDateTimeBestEffortUSOrNull—  
2021-02-11 00:12:57 |
```

Запрос:

```
SELECT parseDateTimeBestEffortUSOrNull('02.10.2021') AS parseDateTimeBestEffortUSOrNull;
```

Результат:

```
parseDateTimeBestEffortUSOrNull—  
2021-02-10 00:00:00 |
```

Запрос:

```
SELECT parseDateTimeBestEffortUSOrNull('10.2021') AS parseDateTimeBestEffortUSOrNull;
```

Результат:

```
parseDateTimeBestEffortUSOrNull—
```

```
NULL |
```

## parseDateTimeBestEffortUSOrZero

Работает аналогично функции [parseDateTimeBestEffortUS](#), но в отличие от нее возвращает нулевую дату (1970-01-01) или нулевую дату со временем (1970-01-01 00:00:00), если входная строка не может быть преобразована в тип данных [DateTime](#).

### Синтаксис

```
parseDateTimeBestEffortUSOrZero(time_string[, time_zone])
```

### Аргументы

- `time_string` — строка, содержащая дату или дату со временем для преобразования. Дата должна быть в американском формате (MM/DD/YYYY и т.д.). [String](#).
- `time_zone` — [часовой пояс](#). Функция анализирует `time_string` в соответствии с заданным часовым поясом. Опциональный параметр. [String](#).

### Поддерживаемые нестандартные форматы

- Страна в формате [unix timestamp](#), содержащая 9-10 цифр.
- Страна, содержащая дату и время: YYYYMMDDhhmmss, MM/DD/YYYY hh:mm:ss, MM-DD-YY hh:mm, YYYY-MM-DD hh:mm:ss и т.д.
- Страна, содержащая дату без времени: YYYY, YYYYMM, YYYY\*MM, MM/DD/YYYY, MM-DD-YY и т.д.
- Страна, содержащая день и время: DD, DD hh, DD hh:mm. В этом случае YYYY-MM заменяется на 2000-01.
- Страна, содержащая дату и время, а также информацию о часовом поясе: YYYY-MM-DD hh:mm:ss ±hh:mm и т.д. Например, 2020-12-12 17:36:00 -5:00.

### Возвращаемые значения

- `time_string`, преобразованная в тип данных [DateTime](#).
- Нулевая дата или нулевая дата со временем, если входная строка не может быть преобразована в тип данных [DateTime](#).

### Примеры

Запрос:

```
SELECT parseDateTimeBestEffortUSOrZero('02/10/2021 21:12:57') AS parseDateTimeBestEffortUSOrZero;
```

Результат:

```
parseDateTimeBestEffortUSOrZero—
```

```
2021-02-10 21:12:57 |
```

Запрос:

```
SELECT parseDateTimeBestEffortUSOrZero('02-10-2021 21:12:57 GMT', 'Europe/Moscow') AS  
parseDateTimeBestEffortUSOrZero;
```

Результат:

```
parseDateTimeBestEffortUSOrZero—  
2021-02-11 00:12:57 |
```

Запрос:

```
SELECT parseDateTimeBestEffortUSOrZero('02.10.2021') AS parseDateTimeBestEffortUSOrZero;
```

Результат:

```
parseDateTimeBestEffortUSOrZero—  
2021-02-10 00:00:00 |
```

Запрос:

```
SELECT parseDateTimeBestEffortUSOrZero('02.2021') AS parseDateTimeBestEffortUSOrZero;
```

Результат:

```
parseDateTimeBestEffortUSOrZero—  
1970-01-01 00:00:00 |
```

## parseDateTime64BestEffort

Работает аналогично функции [parseDateTimeBestEffort](#), но также принимает миллисекунды и микросекунды. Возвращает тип данных [DateTime](#).

### Синтаксис

```
parseDateTime64BestEffort(time_string [, precision [, time_zone]])
```

### Аргументы

- `time_string` — строка, содержащая дату или дату со временем, которые нужно преобразовать. [String](#).
- `precision` — требуемая точность: 3 — для миллисекунд, 6 — для микросекунд. По умолчанию — 3. Необязательный. [UInt8](#).
- `time_zone` — [Timezone](#). Разбирает значение `time_string` в зависимости от часового пояса. Необязательный. [String](#).

### Возвращаемое значение

- `time_string`, преобразованная в тип данных [DateTime](#).

### Примеры

Запрос:

```
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;
```

Результат:

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

Работает аналогично функции [parseDateTime64BestEffort](#), но возвращает `NULL`, если формат даты не может быть обработан.

## parseDateTime64BestEffortOrZero

Работает аналогично функции [parseDateTime64BestEffort](#), но возвращает нулевую дату и время, если формат даты не может быть обработан.

## toLowCardinality

Преобразует входные данные в версию [LowCardianlity](#) того же типа данных.

Чтобы преобразовать данные из типа `LowCardinality`, используйте функцию [CAST](#). Например, `CAST(x as String)`.

### Синтаксис

```
toLowCardinality(expr)
```

### Аргументы

- `expr` — выражение, которое в результате преобразуется в один из [поддерживаемых типов данных](#).

### Возвращаемое значение

- Результат преобразования `expr`.

Тип: `LowCardinality(expr_result_type)`

### Пример

Запрос:

```
SELECT toLowCardinality('1');
```

Результат:

```
└─toLowCardinality('1')─  
  └─1
```

`toUnixTimestamp64Milli`

`toUnixTimestamp64Micro`

`toUnixTimestamp64Nano`

Преобразует значение `DateTime64` в значение `Int64` с фиксированной точностью менее одной секунды.

Входное значение округляется соответствующим образом вверх или вниз в зависимости от его точности.

## Примечание

Возвращаемое значение — это временная метка в UTC, а не в часовом поясе `DateTime64`.

## Синтаксис

```
toUnixTimestamp64Milli(value)
```

## Аргументы

- `value` — значение `DateTime64` с любой точностью.

## Возвращаемое значение

- Значение `value`, преобразованное в тип данных `Int64`.

## Примеры

Запрос:

```
WITH toDateTime64('2019-09-16 19:20:12.345678910', 6) AS dt64  
SELECT toUnixTimestamp64Milli(dt64);
```

Результат:

```
└─toUnixTimestamp64Milli(dt64)─  
  └─1568650812345 |
```

Запрос:

```
WITH toDateTime64('2019-09-16 19:20:12.345678910', 6) AS dt64  
SELECT toUnixTimestamp64Nano(dt64);
```

Результат:

```
└─toUnixTimestamp64Nano(dt64)─  
  └─1568650812345678000 |
```

## fromUnixTimestamp64Milli fromUnixTimestamp64Micro fromUnixTimestamp64Nano

Преобразует значение Int64 в значение DateTime64 с фиксированной точностью менее одной секунды и дополнительным часовым поясом. Входное значение округляется соответствующим образом вверх или вниз в зависимости от его точности. Обратите внимание, что входное значение обрабатывается как метка времени UTC, а не метка времени в заданном (или неявном) часовом поясе.

### Синтаксис

```
fromUnixTimestamp64Milli(value [, ti])
```

### Аргументы

- `value` — значение типа Int64 с любой точностью.
- `timezone` — (не обязательный параметр) часовой пояс в формате String для возвращаемого результата.

### Возвращаемое значение

- Значение `value`, преобразованное в тип данных DateTime64.

### Пример

Запрос:

```
WITH CAST(1234567891011, 'Int64') AS i64
SELECT fromUnixTimestamp64Milli(i64, 'UTC');
```

Результат:

```
fromUnixTimestamp64Milli(i64, 'UTC')—
2009-02-13 23:31:31.011 |
```

## formatRow

Преобразует произвольные выражения в строку заданного формата.

### Синтаксис

```
formatRow(format, x, y, ...)
```

### Аргументы

- `format` — текстовый формат. Например, [CSV](#), [TSV](#).
- `x,y, ...` — выражения.

### Возвращаемое значение

- Отформатированная строка (в текстовых форматах обычно с завершающим переводом строки).

## Пример

Запрос:

```
SELECT formatRow('CSV', number, 'good')
FROM numbers(3);
```

Результат:

```
formatRow('CSV', number, 'good')—
0,"good"
|
1,"good"
|
2,"good"
```

## formatRowNoNewline

Преобразует произвольные выражения в строку заданного формата. При этом удаляет лишние переводы строк `\n`, если они появились.

### Синтаксис

```
formatRowNoNewline(format, x, y, ...)
```

### Аргументы

- `format` — текстовый формат. Например, [CSV](#), [TSV](#).
- `x,y, ...` — выражения.

### Возвращаемое значение

- Отформатированная строка (в текстовых форматах без завершающего перевода строки).

## Пример

Запрос:

```
SELECT formatRowNoNewline('CSV', number, 'good')
FROM numbers(3);
```

Результат:

```
formatRowNoNewline('CSV', number, 'good')—
0,"good"
1,"good"
2,"good"
```

## snowflakeToDateTime

Извлекает время из [Snowflake ID](#) в формате [DateTime](#).

### Синтаксис

```
snowflakeToDateTime(value [, time_zone])
```

## Аргументы

- `value` — Snowflake ID. [Int64](#).
- `time_zone` — [временная зона сервера](#). Функция распознает `time_string` в соответствии с часовым поясом. Необязательный. [String](#).

## Возвращаемое значение

- Значение, преобразованное в формат [DateTime](#).

## Пример

Запрос:

```
SELECT snowflakeToDateTime(CAST('1426860702823350272', 'Int64'), 'UTC');
```

Результат:

```
└─snowflakeToDateTime(CAST('1426860702823350272', 'Int64'), 'UTC')─┐  
   2021-08-15 10:57:56 |
```

## snowflakeToDateTime64

Извлекает время из [Snowflake ID](#) в формате [DateTime64](#).

## Синтаксис

```
snowflakeToDateTime64(value [, time_zone])
```

## Аргументы

- `value` — Snowflake ID. [Int64](#).
- `time_zone` — [временная зона сервера](#). Функция распознает `time_string` в соответствии с часовым поясом. Необязательный. [String](#).

## Возвращаемое значение

- Значение, преобразованное в формат [DateTime64](#).

## Пример

Запрос:

```
SELECT snowflakeToDateTime64(CAST('1426860802823350272', 'Int64'), 'UTC');
```

Результат:

```
└─snowflakeToDateTime64(CAST('1426860802823350272', 'Int64'), 'UTC')─┐  
   2021-08-15 10:58:19.841 |
```

## dateTimeToSnowflake

Преобразует значение **DateTime** в первый идентификатор **Snowflake ID** на текущий момент.

### Syntax

```
dateTimeToSnowflake(value)
```

### Аргументы

- `value` — дата и время. **DateTime**.

### Возвращаемое значение

- Значение, преобразованное в **Int64**, как первый идентификатор Snowflake ID в момент выполнения.

### Пример

Запрос:

```
WITH toDateTime('2021-08-15 18:57:56', 'Asia/Shanghai') AS dt SELECT dateTimeToSnowflake(dt);
```

Результат:

```
dateTimeToSnowflake(dt)  
1426860702823350272
```

## dateTime64ToSnowflake

Преобразует значение **DateTime64** в первый идентификатор **Snowflake ID** на текущий момент.

### Синтаксис

```
dateTime64ToSnowflake(value)
```

### Аргументы

- `value` — дата и время. **DateTime64**.

### Возвращаемое значение

- Значение, преобразованное в **Int64**, как первый идентификатор Snowflake ID в момент выполнения.

### Пример

Запрос:

```
WITH toDateTime64('2021-08-15 18:57:56.492', 3, 'Asia/Shanghai') AS dt64 SELECT dateTime64ToSnowflake(dt64);
```

Результат:

```
dateTime64ToSnowflake(dt64)
1426860704886947840 |
```

## ФУНКЦИИ ДЛЯ РАБОТЫ С ДАТАМИ И ВРЕМЕНЕМ

### Поддержка часовых поясов

Все функции по работе с датой и временем, для которых это имеет смысл, могут принимать второй, необязательный аргумент - имя часового пояса. Пример: Asia/Yekaterinburg. В этом случае, они используют не локальный часовой пояс (по умолчанию), а указанный.

```
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

Возвращает часовой пояс сервера.

Если функция вызывается в контексте распределенной таблицы, то она генерирует обычный столбец со значениями, актуальными для каждого шарда. Иначе возвращается константа.

### Синтаксис

```
timeZone()
```

Синоним: timezone.

### Возвращаемое значение

- Часовой пояс.

Тип: **String**.

## toTimeZone

Переводит дату или дату с временем в указанный часовой пояс. Часовой пояс - это атрибут типов Date и DateTime. Внутреннее значение (количество секунд) поля таблицы или результирующего столбца не изменяется, изменяется тип поля и, соответственно, его текстовое отображение.

### Синтаксис

```
toTimezone(value, timezone)
```

Синоним: **toTimezone**.

### Аргументы

- value** — время или дата с временем. **DateTime64**.

- `timezone` — часовой пояс для возвращаемого значения. [String](#).

## Возвращаемое значение

- Дата с временем.

Тип: [DateTime](#).

## Пример

Запрос:

```
SELECT toDateTime('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;
```

Результат:

```
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')` изменяет тип `DateTime('UTC')` в `DateTime('Asia/Yekaterinburg')`. Значение (unix-время) 1546300800 остается неизменным, но текстовое отображение (результат функции `toString()`) меняется `time_utc: 2019-01-01 00:00:00` в `time_yekat: 2019-01-01 05:00:00`.

## timeZoneOf

Возвращает название часового пояса для значений типа [DateTime](#) и [DateTime64](#).

## Синтаксис

```
timeZoneOf(value)
```

Синоним: `timezoneOf`.

## Аргументы

- `value` — Дата с временем. [DateTime](#) или [DateTime64](#).

## Возвращаемое значение

- Название часового пояса.

Тип: [String](#).

## Пример

Запрос:

```
SELECT timezoneOf(now());
```

Результат:

```
timezoneOf(now())  
Etc/UTC
```

## timeZoneOffset

Возвращает смещение часового пояса в секундах от [UTC](#). Функция учитывает [летнее время](#) и исторические изменения часовых поясов, которые действовали на указанную дату.

Для вычисления смещения используется информация из [базы данных IANA](#).

### Синтаксис

```
timeZoneOffset(value)
```

Синоним: `timezoneOffset`.

### Аргументы

- `value` — Дата с временем. [DateTime](#) or [DateTime64](#).

### Возвращаемое значение

- Смещение в секундах от UTC.

Тип: [Int32](#).

### Пример

Запрос:

```
SELECT toDate('2021-04-21 10:20:30', 'Europe/Moscow') AS Time, typeName(Time) AS Type,  
timeZoneOffset(Time) AS Offset_in_seconds, (Offset_in_seconds / 3600) AS Offset_in_hours;
```

Результат:

Time	Type	Offset_in_seconds	Offset_in_hours
2021-04-21 10:20:30	DateTime('Europe/Moscow')	10800	3

## toYear

Переводит дату или дату-с-временем в число типа `UInt16`, содержащее номер года (AD).

Синоним: `YEAR`.

## toQuarter

Переводит дату или дату-с-временем в число типа `UInt8`, содержащее номер квартала.

Синоним: `QUARTER`.

## toMonth

Переводит дату или дату-с-временем в число типа UInt8, содержащее номер месяца (1-12).

Синоним: MONTH.

## toDayOfYear

Переводит дату или дату-с-временем в число типа UInt16, содержащее номер дня года (1-366).

Синоним: DAYOFYEAR.

## toDayOfMonth

Переводит дату или дату-с-временем в число типа UInt8, содержащее номер дня в месяце (1-31).

Синонимы: DAYOFMONTH, DAY.

## toDayOfWeek

Переводит дату или дату-с-временем в число типа UInt8, содержащее номер дня в неделе (понедельник - 1, воскресенье - 7).

Синоним: DAYOFWEEK.

## toHour

Переводит дату-с-временем в число типа UInt8, содержащее номер часа в сутках (0-23).

Функция исходит из допущения, что перевод стрелок вперёд, если осуществляется, то на час, в два часа ночи, а перевод стрелок назад, если осуществляется, то на час, в три часа ночи (что, в общем, не верно - даже в Москве два раза перевод стрелок был осуществлён в другое время).

Синоним: HOUR.

## toMinute

Переводит дату-с-временем в число типа UInt8, содержащее номер минуты в часе (0-59).

Синоним: MINUTE.

## toSecond

Переводит дату-с-временем в число типа UInt8, содержащее номер секунды в минуте (0-59).

Секунды координации не учитываются.

Синоним: SECOND.

## toUnixTimestamp

Переводит дату-с-временем в число типа UInt32 -- Unix Timestamp ([https://en.wikipedia.org/wiki/Unix\\_time](https://en.wikipedia.org/wiki/Unix_time)).

Для аргумента String, строка конвертируется в дату и время в соответствии с часовым поясом (необязательный второй аргумент, часовой пояс сервера используется по умолчанию).

### Синтаксис

```
toUnixTimestamp(datetime)
toUnixTimestamp(str, [timezone])
```

## **Возвращаемое значение**

- Возвращает Unix Timestamp.

Тип: UInt32.

## **Пример**

Запрос:

```
SELECT toUnixTimestamp('2017-11-05 08:07:47', 'Asia/Tokyo') AS unix_timestamp;
```

Результат:

```
unix_timestamp  
1509836867 |
```

## **Attention**

Date или DateTime это возвращаемый тип функций toStartOf\*, который описан ниже. Несмотря на то, что эти функции могут принимать DateTime64 в качестве аргумента, если переданное значение типа DateTime64 выходит за пределы нормального диапазона (с 1925 по 2283 год), то это даст неверный результат.

## **toStartOfYear**

Округляет дату или дату-с-временем вниз до первого дня года.

Возвращается дата.

## **toStartOfISOYear**

Округляет дату или дату-с-временем вниз до первого дня ISO года. Возвращается дата.

Начало ISO года отличается от начала обычного года, потому что в соответствии с ISO 8601:1988 первая неделя года - это неделя с четырьмя или более днями в этом году.

1 Января 2017 г. - воскресение, т.е. первая ISO неделя 2017 года началась в понедельник 2 января, поэтому 1 января 2017 это 2016 ISO-год, который начался 2016-01-04.

```
SELECT toStartOfISOYear(toDate('2017-01-01')) AS ISOYear20170101;
```

```
ISOYear20170101  
2016-01-04 |
```

## **toStartOfQuarter**

Округляет дату или дату-с-временем вниз до первого дня квартала.

Первый день квартала - это одно из 1 января, 1 апреля, 1 июля, 1 октября.

Возвращается дата.

## **toStartOfMonth**

Округляет дату или дату-с-временем вниз до первого дня месяца.  
Возвращается дата.

## Attention

Возвращаемое значение для некорректных дат зависит от реализации. ClickHouse может вернуть нулевую дату, выбросить исключение, или выполнить «естественное» перетекание дат между месяцами.

## toMonday

Округляет дату или дату-с-временем вниз до ближайшего понедельника.  
Возвращается дата.

## toStartOfWeek(t[,mode])

Округляет дату или дату со временем до ближайшего воскресенья или понедельника в соответствии с mode.

Возвращается дата.

Аргумент mode работает точно так же, как аргумент mode [toWeek\(\)](#). Если аргумент mode опущен, то используется режим 0.

## toStartOfDay

Округляет дату-с-временем вниз до начала дня. Возвращается дата-с-временем.

## toStartOfHour

Округляет дату-с-временем вниз до начала часа.

## toStartOfMinute

Округляет дату-с-временем вниз до начала минуты.

## toStartOfSecond

Отсекает доли секунды.

### Синтаксис

```
toStartOfSecond(value, [timezone])
```

### Аргументы

- `value` — дата и время. [DateTime64](#).
- `timezone` — [часовой пояс](#) для возвращаемого значения (необязательно). Если параметр не задан, используется часовой пояс параметра `value`. [String](#).

### Возвращаемое значение

- Входное значение с отсеченными долями секунды.

Тип: [DateTime64](#).

### Примеры

Пример без часового пояса:

```
WITH toDateTime64('2020-01-01 10:20:30.999', 3) AS dt64 SELECT toStartOfSecond(dt64);
```

Результат:

```
toStartOfSecond(dt64)  
2020-01-01 10:20:30.000 |
```

Пример с часовым поясом:

```
WITH toDateTime64('2020-01-01 10:20:30.999', 3) AS dt64 SELECT toStartOfSecond(dt64, 'Europe/Moscow');
```

Результат:

```
toStartOfSecond(dt64, 'Europe/Moscow')  
2020-01-01 13:20:30.000 |
```

#### Смотрите также

- Часовая зона сервера, конфигурационный параметр [timezone](#).

## toStartOfFiveMinute

Округляет дату-с-временем вниз до начала пятиминутного интервала.

## toStartOfTenMinutes

Округляет дату-с-временем вниз до начала десятиминутного интервала.

## toStartOfFifteenMinutes

Округляет дату-с-временем вниз до начала пятнадцатиминутного интервала.

## toStartOfInterval(time\_or\_data, INTERVAL x unit [, time\_zone])

Обобщение остальных функций `toStartOf*`. Например,  
`toStartOfInterval(t, INTERVAL 1 year)` возвращает то же самое, что и `toStartOfYear(t)`,  
`toStartOfInterval(t, INTERVAL 1 month)` возвращает то же самое, что и `toStartOfMonth(t)`,  
`toStartOfInterval(t, INTERVAL 1 day)` возвращает то же самое, что и `toStartOfDay(t)`,  
`toStartOfInterval(t, INTERVAL 15 minute)` возвращает то же самое, что и `toStartOfFifteenMinutes(t)`, и т.п.

## toTime

Переводит дату-с-временем на некоторую фиксированную дату, сохраняя при этом время.

## toRelativeYearNum

Переводит дату-с-временем или дату в номер года, начиная с некоторого фиксированного момента в прошлом.

## toRelativeQuarterNum

Переводит дату-с-временем или дату в номер квартала, начиная с некоторого фиксированного момента в прошлом.

## toRelativeMonthNum

Переводит дату-с-временем или дату в номер месяца, начиная с некоторого фиксированного момента в прошлом.

## toRelativeWeekNum

Переводит дату-с-временем или дату в номер недели, начиная с некоторого фиксированного момента в прошлом.

## toRelativeDayNum

Переводит дату-с-временем или дату в номер дня, начиная с некоторого фиксированного момента в прошлом.

## toRelativeHourNum

Переводит дату-с-временем в номер часа, начиная с некоторого фиксированного момента в прошлом.

## toRelativeMinuteNum

Переводит дату-с-временем в номер минуты, начиная с некоторого фиксированного момента в прошлом.

## toRelativeSecondNum

Переводит дату-с-временем в номер секунды, начиная с некоторого фиксированного момента в прошлом.

## toISOYear

Переводит дату-с-временем или дату в число типа UInt16, содержащее номер ISO года. ISO год отличается от обычного года, потому что в соответствии с [ISO 8601:1988](#) ISO год начинается необязательно первого января.

### Пример

Запрос:

```
SELECT
    toDate('2017-01-01') AS date,
    toYear(date),
    toISOYear(date)
```

Результат:

date	toYear(toDate('2017-01-01'))	toISOYear(toDate('2017-01-01'))
2017-01-01	2017	2016

## toISOWeek

Переводит дату-с-временем или дату в число типа UInt8, содержащее номер ISO недели. Начало ISO года отличается от начала обычного года, потому что в соответствии с [ISO 8601:1988](#) первая неделя года - это неделя с четырьмя или более днями в этом году.

1 Января 2017 г. - воскресение, т.е. первая ISO неделя 2017 года началась в понедельник 2 января, поэтому 1 января 2017 это последняя неделя 2016 года.

## Пример

Запрос:

```
SELECT
    toISOWeek(toDate('2017-01-01')) AS ISOWeek20170101,
    toISOWeek(toDate('2017-01-02')) AS ISOWeek20170102
```

Результат:

```
ISOWeek20170101 ISOWeek20170102
52 | 1 |
```

## toWeek(date[, mode][, timezone])

Переводит дату-с-временем или дату в число UInt8, содержащее номер недели. Второй аргументом mode задает режим, начинается ли неделя с воскресенья или с понедельника и должно ли возвращаемое значение находиться в диапазоне от 0 до 53 или от 1 до 53. Если аргумент mode опущен, то используется режим 0.

toISOWeek() эквивалентно toWeek(date,3).

Описание режимов (mode):

Mode	Первый день недели	Диапазон	Неделя 1 это первая неделя ...
0	Воскресенье	0-53	с воскресеньем в этом году
1	Понедельник	0-53	с 4-мя или более днями в этом году
2	Воскресенье	1-53	с воскресеньем в этом году
3	Понедельник	1-53	с 4-мя или более днями в этом году
4	Воскресенье	0-53	с 4-мя или более днями в этом году
5	Понедельник	0-53	с понедельником в этом году
6	Воскресенье	1-53	с 4-мя или более днями в этом году
7	Понедельник	1-53	с понедельником в этом году
8	Воскресенье	1-53	содержащая 1 января
9	Понедельник	1-53	содержащая 1 января

Для режимов со значением «с 4 или более днями в этом году» недели нумеруются в соответствии с ISO 8601:1988:

- Если неделя, содержащая 1 января, имеет 4 или более дней в новом году, это неделя 1.
- В противном случае это последняя неделя предыдущего года, а следующая неделя - неделя 1.

Для режимов со значением «содержит 1 января», неделя 1 – это неделя содержащая 1 января. Не имеет значения, сколько дней в новом году содержала неделя, даже если она содержала только один день.

## Пример

Запрос:

```
SELECT toDate('2016-12-27') AS date, toWeek(date) AS week0, toWeek(date,1) AS week1, toWeek(date,9) AS week9;
```

Результат:

date	week0	week1	week9
2016-12-27	52	52	1

## toYearWeek(date[,mode])

Возвращает год и неделю для даты. Год в результате может отличаться от года в аргументе даты для первой и последней недели года.

Аргумент mode работает точно так же, как аргумент mode [toWeek\(\)](#). Если mode не задан, используется режим 0.

[toISOYear\(\)](#) эквивалентно `intDiv(toYearWeek(date,3),100)`.

## Пример

Запрос:

```
SELECT toDate('2016-12-27') AS date, toYearWeek(date) AS yearWeek0, toYearWeek(date,1) AS yearWeek1, toYearWeek(date,9) AS yearWeek9;
```

Результат:

date	yearWeek0	yearWeek1	yearWeek9
2016-12-27	201652	201652	201701

## date\_trunc

Отсекает от даты и времени части, меньшие чем указанная часть.

### Синтаксис

```
date_trunc(unit, value[, timezone])
```

Синоним: `dateTrunc`.

### Аргументы

- `unit` — единица измерения времени, в которой задана отсекаемая часть. [String Literal](#).

Возможные значения:

- `second`
  - `minute`
  - `hour`
  - `day`
  - `week`
  - `month`
  - `quarter`
  - `year`
- `value` — дата и время. [DateTime](#) или [DateTime64](#).
  - `timezone` — [часовой пояс](#) для возвращаемого значения (необязательно). Если параметр не задан, используется часовой пояс параметра `value`. [String](#)

## Возвращаемое значение

- Дата и время, отсеченные до указанной части.

Тип: [Datetime](#).

## Примеры

Запрос без указания часового пояса:

```
SELECT now(), date_trunc('hour', now());
```

Результат:

now()	date_trunc('hour', now())
2020-09-28 10:40:45	2020-09-28 10:00:00

Запрос с указанием часового пояса:

```
SELECT now(), date_trunc('hour', now(), 'Europe/Moscow');
```

Результат:

now()	date_trunc('hour', now(), 'Europe/Moscow')
2020-09-28 10:46:26	2020-09-28 13:00:00

## Смотрите также

- [toStartOfInterval](#)

## date\_add

Добавляет интервал времени или даты к указанной дате или дате со временем.

## Синтаксис

```
date_add(unit, value, date)
```

Синонимы: dateAdd, DATE\_ADD.

## Аргументы

- `unit` — единица измерения времени, в которой задан интервал для добавления. [String](#).  
Возможные значения:
  - `second`
  - `minute`
  - `hour`
  - `day`
  - `week`
  - `month`
  - `quarter`
  - `year`
- `value` — значение интервала для добавления. [Int](#).
- `date` — дата или дата со временем, к которой добавляется `value`. [Date](#) или [DateTime](#).

## Возвращаемое значение

Дата или дата со временем, полученная в результате добавления `value`, выраженного в `unit`, к `date`.

Тип: [Date](#) или [DateTime](#).

## Пример

Запрос:

```
SELECT date_add(YEAR, 3, toDate('2018-01-01'));
```

Результат:

```
plus(toDate('2018-01-01'), toIntervalYear(3))-->  
2021-01-01 |
```

## date\_diff

Вычисляет разницу между двумя значениями дат или дат со временем.

## Синтаксис

```
date_diff('unit', startdate, enddate, [timezone])
```

Синонимы: dateDiff, DATE\_DIFF.

## Аргументы

- `unit` — единица измерения времени, в которой будет выражено возвращаемое значение функции. **String**.  
Возможные значения:
  - `second`
  - `minute`
  - `hour`
  - `day`
  - `week`
  - `month`
  - `quarter`
  - `year`
- `startdate` — первая дата или дата со временем, которая вычитается из `enddate`. **Date** или **DateTime**.
- `enddate` — вторая дата или дата со временем, из которой вычитается `startdate`. **Date** или **DateTime**.
- `timezone` — **часовой пояс** (необязательно). Если этот аргумент указан, то он применяется как для `startdate`, так и для `enddate`. Если этот аргумент не указан, то используются часовые пояса аргументов `startdate` и `enddate`. Если часовые пояса аргументов `startdate` и `enddate` не совпадают, то результат не определен. **String**.

## Возвращаемое значение

Разница между `enddate` и `startdate`, выраженная в `unit`.

Тип: **Int**.

## Пример

Запрос:

```
SELECT dateDiff('hour', toDateTime('2018-01-01 22:00:00'), toDateTime('2018-01-02 23:00:00'));
```

Результат:

```
dateDiff('hour', toDateTime('2018-01-01 22:00:00'), toDateTime('2018-01-02 23:00:00'))—  
25 |
```

## date\_sub

Вычитает интервал времени или даты из указанной даты или даты со временем.

## Синтаксис

```
date_sub(unit, value, date)
```

Синонимы: `dateSub`, `DATE_SUB`.

## Аргументы

- `unit` — единица измерения времени, в которой задан интервал для вычитания. [String](#).

Возможные значения:

- `second`
- `minute`
- `hour`
- `day`
- `week`
- `month`
- `quarter`
- `year`
- `value` — значение интервала для вычитания. [Int](#).
- `date` — дата или дата со временем, из которой вычитается `value`. [Date](#) или [DateTime](#).

## Возвращаемое значение

Дата или дата со временем, полученная в результате вычитания `value`, выраженного в `unit`, из `date`.

Тип: [Date](#) или [DateTime](#).

## Пример

Запрос:

```
SELECT date_sub(YEAR, 3, toDate('2018-01-01'));
```

Результат:

```
minus(toDate('2018-01-01'), toIntervalYear(3))|  
2015-01-01 |
```

## timestamp\_add

Добавляет интервал времени к указанной дате или дате со временем.

## Синтаксис

```
timestamp_add(date, INTERVAL value unit)
```

Синонимы: `timeStampAdd`, `TIMESTAMP_ADD`.

## Аргументы

- `date` — дата или дата со временем. [Date](#) или [DateTime](#).
- `value` — значение интервала для добавления. [Int](#).

- `unit` — единица измерения времени, в которой задан интервал для добавления. [String](#).

Возможные значения:

- `second`
- `minute`
- `hour`
- `day`
- `week`
- `month`
- `quarter`
- `year`

## Возвращаемое значение

Дата или дата со временем, полученная в результате добавления `value`, выраженного в `unit`, к `date`.

Тип: [Date](#) или [DateTime](#).

## Пример

Запрос:

```
select timestamp_add(toDate('2018-01-01'), INTERVAL 3 MONTH);
```

Результат:

```
plus(toDate('2018-01-01'), toIntervalMonth(3))—  
2018-04-01 |
```

## timestamp\_sub

Вычитает интервал времени из указанной даты или даты со временем.

## Синтаксис

```
timestamp_sub(unit, value, date)
```

Синонимы: `timeStampSub`, `TIMESTAMP_SUB`.

## Аргументы

- `unit` — единица измерения времени, в которой задан интервал для вычитания. [String](#).

Возможные значения:

- `second`
  - `minute`
  - `hour`
  - `day`
  - `week`
  - `month`
  - `quarter`
  - `year`
- `value` — значение интервала для вычитания. [Int](#).
  - `date` — дата или дата со временем. [Date](#) или [DateTime](#).

### Возвращаемое значение

Дата или дата со временем, полученная в результате вычитания `value`, выраженного в `unit`, из `date`.

Тип: [Date](#) или [DateTime](#).

### Пример

Запрос:

```
select timestamp_sub(MONTH, 5, toDate('2018-12-18 01:02:03'));
```

Результат:

```
minus(toDate('2018-12-18 01:02:03'), toIntervalMonth(5))—  
2018-07-18 01:02:03 |
```

## NOW

Возвращает текущую дату и время.

### Синтаксис

```
now([timezone])
```

### Параметры

- `timezone` — [часовой пояс](#) для возвращаемого значения (необязательно). [String](#).

### Возвращаемое значение

- Текущие дата и время.

Тип: [Datetime](#).

### Пример

Запрос без указания часового пояса:

```
SELECT now();
```

Результат:

```
now()  
2020-10-17 07:42:09 |
```

Запрос с указанием часового пояса:

```
SELECT now('Europe/Moscow');
```

Результат:

```
now('Europe/Moscow')  
2020-10-17 10:42:23 |
```

## today

Возвращает текущую дату на момент выполнения запроса. Функция не требует аргументов.  
То же самое, что toDate(now())

## yesterday

Возвращает вчерашнюю дату на момент выполнения запроса.  
Делает то же самое, что today() - 1. Функция не требует аргументов.

## timeSlot

Округляет время до получаса.

Эта функция является специфичной для Яндекс.Метрики, так как пол часа - минимальное время, для которого, если соседние по времени хиты одного посетителя на одном счётчике отстоят друг от друга строго более, чем на это время, визит может быть разбит на два визита. То есть, кортежи (номер счётчика, идентификатор посетителя, тайм-слот) могут использоваться для поиска хитов, входящий в соответствующий визит.

## timeSlots(StartTime, Duration[, Size])

Для интервала времени, начинающегося в 'StartTime' и продолжающегося 'Duration' секунд, возвращает массив моментов времени, состоящий из округлений вниз до 'Size' точек в секундах из этого интервала. 'Size' - необязательный параметр, константный UInt32, по умолчанию равен 1800.

Например, `timeSlots(toDateTime('2012-01-01 12:20:00'), toUInt32(600)) = [toDateTime('2012-01-01 12:00:00'), toDateTime('2012-01-01 12:30:00')]`.

Это нужно для поиска хитов, входящих в соответствующий визит.

## toYYYYMM

Переводит дату или дату со временем в число типа UInt32, содержащее номер года и месяца (YYYY \* 100 + MM).

## toYYYYMMDD

Переводит дату или дату со временем в число типа UInt32, содержащее номер года, месяца и дня (YYYY \* 10000 + MM \* 100 + DD).

## toYYYYMMDDhhmmss

Переводит дату или дату со временем в число типа UInt64 содержащее номер года, месяца, дня и время (YYYY \* 10000000000 + MM \* 100000000 + DD \* 1000000 + hh \* 10000 + mm \* 100 + ss).

## formatDateTime

Функция преобразует дату-и-время в строку по заданному шаблону. Важно: шаблон — константное выражение, поэтому использовать разные шаблоны в одной колонке не получится.

### Синтаксис

```
formatDateTime(Time, Format[, Timezone\])
```

### Возвращаемое значение

Возвращает значение времени и даты в определенном вами формате.

### Поля подстановки

Используйте поля подстановки для того, чтобы определить шаблон для выводимой строки. В колонке «Пример» результат работы функции для времени 2018-01-02 22:33:44.

Поле	Описание	Пример
%C	номер года, поделённый на 100 (00-99)	20
%d	день месяца, с ведущим нулём (01-31)	02
%D	короткая запись %m/%d/%y	01/02/18
%e	день месяца, с ведущим пробелом ( 1-31)	2
%F	короткая запись %Y-%m-%d	2018-01-02
%G	четырехзначный формат вывода ISO-года, который основывается на особом подсчете номера недели согласно <a href="#">стандарту ISO 8601</a> , обычно используется вместе с %V	2018
%g	двухзначный формат вывода года по стандарту ISO 8601	18
%H	час в 24-часовом формате (00-23)	22
%I	час в 12-часовом формате (01-12)	10
%j	номер дня в году, с ведущими нулями (001-366)	002
%m	месяц, с ведущим нулём (01-12)	01
%M	минуты, с ведущим нулём (00-59)	33

Поле	Описание	Пример
%n	символ переноса строки ('')	
%p	обозначения AM или PM	PM
%Q	квартал (1-4)	1
%R	короткая запись %H:%M	22:33
%S	секунды, с ведущими нулями (00-59)	44
%t	символ табуляции (')	
%T	формат времени ISO 8601, одинаковый с %H:%M:%S	22:33:44
%u	номер дня недели согласно ISO 8601, понедельник - 1, воскресенье - 7	2
%V	номер недели согласно ISO 8601 (01-53)	01
%w	номер дня недели, начиная с воскресенья (0-6)	2
%y	год, последние 2 цифры (00-99)	18
%Y	год, 4 цифры	2018
%%	символ %	%

## Пример

Запрос:

```
SELECT formatDateTime(toDate('2010-01-04'), '%g');
```

Результат:

```
formatDateTime(toDate('2010-01-04'), '%g')—  
10 |
```

## dateName

Возвращает указанную часть даты.

### Синтаксис

```
dateName(date_part, date)
```

### Аргументы

- date\_part — часть даты. Возможные значения: 'year', 'quarter', 'month', 'week', 'dayofyear', 'day', 'weekday', 'hour', 'minute', 'second'. **String**.

- `date` — дата. `Date`, `DateTime` или `DateTime64`.
- `timezone` — часовой пояс. Необязательный аргумент. `String`.

## Возвращаемое значение

- Указанная часть даты.

Тип: `String`.

## Пример

Запрос:

```
WITH toDateTime('2021-04-14 11:22:33') AS date_value
SELECT dateName('year', date_value), dateName('month', date_value), dateName('day', date_value);
```

Результат:

dateName('year', date_value)	—	dateName('month', date_value)	—	dateName('day', date_value)	—
2021	April	14			

## FROM\_UNIXTIME

Функция преобразует Unix timestamp в календарную дату и время.

### Примеры

Если указан только один аргумент типа `Integer`, то функция действует так же, как `toDateTime`, и возвращает тип `DateTime`.

Запрос:

```
SELECT FROM_UNIXTIME(423543535);
```

Результат:

FROM_UNIXTIME(423543535)	—	1983-06-04 10:58:55	
--------------------------	---	---------------------	--

В случае, когда есть два аргумента: первый типа `Integer` или `DateTime`, а второй является строкой постоянного формата — функция работает также, как `formatDateTime`, и возвращает значение типа `String`.

Запрос:

```
SELECT FROM_UNIXTIME(1234334543, '%Y-%m-%d %R:%S') AS DateTime;
```

Результат:

DateTime	—	2009-02-11 14:42:23	
----------	---	---------------------	--

# ФУНКЦИИ ДЛЯ РАБОТЫ СО СТРОКАМИ

## empty

Проверяет, является ли входная строка пустой.

### Синтаксис

```
empty(x)
```

Строка считается непустой, если содержит хотя бы один байт, пусть даже это пробел или нулевой байт.

Функция также поддерживает работу с типами [Array](#) и [UUID](#).

### Параметры

- `x` — Входная строка. [String](#).

### Возвращаемое значение

- Возвращает `1` для пустой строки и `0` — для непустой строки.

Тип: [UInt8](#).

### Пример

Запрос:

```
SELECT empty('text');
```

Результат:

```
+-----+  
| empty('text') |  
+-----+  
| 0 |  
+-----+
```

## notEmpty

Проверяет, является ли входная строка непустой.

### Синтаксис

```
notEmpty(x)
```

Строка считается непустой, если содержит хотя бы один байт, пусть даже это пробел или нулевой байт.

Функция также поддерживает работу с типами [Array](#) и [UUID](#).

### Параметры

- `x` — Входная строка. [String](#).

### Возвращаемое значение

- Возвращает `1` для непустой строки и `0` — для пустой строки.

Тип: [UInt8](#).

## Пример

Запрос:

```
SELECT notEmpty('text');
```

Результат:

```
notEmpty('text')  
1 |
```

## length

Возвращает длину строки в байтах (не символах, не кодовых точках).

Тип результата — [UInt64](#).

Функция также работает для массивов.

## lengthUTF8

Возвращает длину строки в кодовых точках Unicode (не символах), при допущении, что строка содержит набор байтов, являющийся текстом в кодировке UTF-8. Если допущение не выполнено, то возвращает какой-нибудь результат (не кидает исключение).

Тип результата — [UInt64](#).

## char\_length, CHAR\_LENGTH

Возвращает длину строки в кодовых точках Unicode (не символах), при допущении, что строка содержит набор байтов, являющийся текстом в кодировке UTF-8. Если допущение не выполнено, возвращает какой-нибудь результат (не кидает исключение).

Тип результата — [UInt64](#).

## character\_length, CHARACTER\_LENGTH

Возвращает длину строки в кодовых точках Unicode (не символах), при допущении, что строка содержит набор байтов, являющийся текстом в кодировке UTF-8. Если допущение не выполнено, возвращает какой-нибудь результат (не кидает исключение).

Тип результата — [UInt64](#).

## leftPad

Дополняет текущую строку слева пробелами или указанной строкой (несколько раз, если необходимо), пока результирующая строка не достигнет заданной длины. Соответствует MySQL функции LPAD.

### Синтаксис

```
leftPad('string', 'length'[, 'pad_string'])
```

### Параметры

- `string` — входная строка, которую необходимо дополнить. [String](#).

- `length` — длина результирующей строки. **UInt**. Если указанное значение меньше, чем длина входной строки, то входная строка возвращается как есть.
- `pad_string` — строка, используемая для дополнения входной строки. **String**. Необязательный параметр. Если не указано, то входная строка дополняется пробелами.

## Возвращаемое значение

- Результирующая строка заданной длины.

Type: **String**.

## Пример

Запрос:

```
SELECT leftPad('abc', 7, '*'), leftPad('def', 7);
```

Результат:

```
+-----+-----+
|leftPad('abc', 7, '*')|leftPad('def', 7)|
|*****abc            |  def          |
+-----+-----+
```

## leftPadUTF8

Дополняет текущую строку слева пробелами или указанной строкой (несколько раз, если необходимо), пока результирующая строка не достигнет заданной длины. Соответствует MySQL функции LPAD. В отличие от функции **leftPad**, измеряет длину строки не в байтах, а в кодовых точках Unicode.

## Синтаксис

```
leftPadUTF8('string','length'[,'pad_string'])
```

## Параметры

- `string` — входная строка, которую необходимо дополнить. **String**.
- `length` — длина результирующей строки. **UInt**. Если указанное значение меньше, чем длина входной строки, то входная строка возвращается как есть.
- `pad_string` — строка, используемая для дополнения входной строки. **String**. Необязательный параметр. Если не указано, то входная строка дополняется пробелами.

## Возвращаемое значение

- Результирующая строка заданной длины.

Type: **String**.

## Пример

Запрос:

```
SELECT leftPadUTF8('абвг', 7, '*'), leftPadUTF8('дежз', 7);
```

Результат:

```
leftPadUTF8('абвг', 7, '*')---leftPadUTF8('дежз', 7)---  
***абвг           |   дежз   |
```

## rightPad

Дополняет текущую строку справа пробелами или указанной строкой (несколько раз, если необходимо), пока результирующая строка не достигнет заданной длины. Соответствует MySQL функции RPAD.

### Синтаксис

```
rightPad('string', 'length'[, 'pad_string'])
```

### Параметры

- `string` — входная строка, которую необходимо дополнить. [String](#).
- `length` — длина результирующей строки. [UInt](#). Если указанное значение меньше, чем длина входной строки, то входная строка возвращается как есть.
- `pad_string` — строка, используемая для дополнения входной строки. [String](#). Необязательный параметр. Если не указано, то входная строка дополняется пробелами.

### Возвращаемое значение

- Результирующая строка заданной длины.

Type: [String](#).

### Пример

Запрос:

```
SELECT rightPad('abc', 7, '*'), rightPad('abc', 7);
```

Результат:

```
rightPad('abc', 7, '*')---rightPad('abc', 7)---  
abc****           | abc     |
```

## rightPadUTF8

Дополняет текущую строку слева пробелами или указанной строкой (несколько раз, если необходимо), пока результирующая строка не достигнет заданной длины. Соответствует MySQL функции RPAD. В отличие от функции [rightPad](#), измеряет длину строки не в байтах, а в кодовых точках Unicode.

### Синтаксис

```
rightPadUTF8('string','length'[, 'pad_string'])
```

### Параметры

- `string` — входная строка, которую необходимо дополнить. [String](#).

- `length` — длина результирующей строки. **UInt**. Если указанное значение меньше, чем длина входной строки, то входная строка возвращается как есть.
- `pad_string` — строка, используемая для дополнения входной строки. **String**. Необязательный параметр. Если не указано, то входная строка дополняется пробелами.

## Возвращаемое значение

- Результирующая строка заданной длины.

Type: **String**.

## Пример

Запрос:

```
SELECT rightPadUTF8('абвг', 7, '*'), rightPadUTF8('абвг', 7);
```

Результат:

```
+-----+-----+
| rightPadUTF8('абвг', 7, '*') | rightPadUTF8('абвг', 7) |
|    абвг***                 |    абвг               |
+-----+-----+
```

## lower, lcase

Переводит ASCII-символы латиницы в строке в нижний регистр.

## upper, ucase

Переводит ASCII-символы латиницы в строке в верхний регистр.

## lowerUTF8

Переводит строку в нижний регистр, при допущении, что строка содержит набор байтов, представляющий текст в кодировке UTF-8.

Не учитывает язык. То есть, для турецкого языка, результат может быть не совсем верным.

Если длина UTF-8 последовательности байтов различна для верхнего и нижнего регистра кодовой точки, то для этой кодовой точки результат работы может быть некорректным.

Если строка содержит набор байтов, не являющийся UTF-8, то поведение не определено.

## upperUTF8

Переводит строку в верхний регистр, при допущении, что строка содержит набор байтов, представляющий текст в кодировке UTF-8.

Не учитывает язык. То есть, для турецкого языка, результат может быть не совсем верным.

Если длина UTF-8 последовательности байтов различна для верхнего и нижнего регистра кодовой точки, то для этой кодовой точки, результат работы может быть некорректным.

Если строка содержит набор байтов, не являющийся UTF-8, то поведение не определено.

## isValidUTF8

Возвращает 1, если набор байтов является корректным в кодировке UTF-8, 0 иначе.

## toValidUTF8

Заменяет некорректные символы UTF-8 на символ ♦ (U+FFFD). Все идущие подряд некорректные символы схлопываются в один заменяющий символ.

```
toValidUTF8(input_string)
```

## Аргументы

- `input_string` — произвольный набор байтов, представленный как объект типа [String](#).

Возвращаемое значение: Корректная строка UTF-8.

## Пример

```
SELECT toValidUTF8('\x61\xF0\x80\x80\x80b');
```

```
toValidUTF8('a♦♦♦♦b')  
a♦b |
```

# repeat

Повторяет строку определенное количество раз и объединяет повторяемые значения в одну строку.

Синоним: [REPEAT](#).

## Синтаксис

```
repeat(s, n)
```

## Аргументы

- `s` — строка для повторения. [String](#).
- `n` — количество повторов. [UInt](#).

## Возвращаемое значение

Строка, состоящая из повторений `n` раз исходной строки `s`. Если `n < 1`, то функция вернет пустую строку.

Тип: [String](#).

## Пример

Запрос:

```
SELECT repeat('abc', 10);
```

Результат:

```
repeat('abc', 10)  
abcabca...abcabcabcabc |
```

# reverse

Разворачивает строку (как последовательность байтов).

## reverseUTF8

Разворачивает последовательность кодовых точек Unicode, при допущении, что строка содержит набор байтов, представляющий текст в кодировке UTF-8. Иначе — что-то делает (не кидает исключение).

## format(pattern, s0, s1, ...)

Форматирует константный шаблон со строками, перечисленными в аргументах. `pattern` — упрощенная версия шаблона в языке Python. Шаблон содержит «заменяющие поля», которые окружены фигурными скобками `{}`. Всё, что не содержится в скобках, интерпретируется как обычный текст и просто копируется. Если нужно использовать символ фигурной скобки, можно экранировать двойной скобкой `{}{ или }}`. Имя полей могут быть числами (нумерация с нуля) или пустыми (тогда они интерпретируются как последовательные числа).

```
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

Склепивает строки, переданные в аргументы, в одну строку без разделителей.

### Синтаксис

```
concat(s1, s2, ...)
```

### Аргументы

Значения типа String или FixedString.

### Возвращаемое значение

Возвращает строку, полученную в результате склейки аргументов.

Если любой из аргументов имеет значение `NULL`, `concat` возвращает значение `NULL`.

### Пример

Запрос:

```
SELECT concat('Hello, ', 'World!');
```

Результат:

```
concat('Hello, ', 'World!')
```

```
Hello, World!
```

## concatAssumeInjective

Аналогична [concat](#). Разница заключается в том, что вам нужно убедиться, что `concat(s1, s2, ...)` → `sn` является инъективным, так как это предположение будет использоваться для оптимизации GROUP BY.

Функция называется «инъективной», если она возвращает разные значения для разных аргументов. Или, иными словами, функция никогда не выдаёт одно и то же значение, если аргументы разные.

### Синтаксис

```
concatAssumeInjective(s1, s2, ...)
```

### Аргументы

Значения типа String или FixedString.

### Возвращаемые значения

Возвращает строку, полученную в результате объединения аргументов.

Если любой из аргументов имеет значение NULL, `concatAssumeInjective` возвращает значение NULL.

### Пример

Вводная таблица:

```
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

Запрос:

```
SELECT concat(key1, key2), sum(value) FROM key_val GROUP BY (key1, key2);
```

Результат:

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)

Возвращает подстроку, начиная с байта по индексу offset, длины length байт. Индексация символов — начиная с единицы (как в стандартном SQL). Аргументы offset и length должны быть константами.

## substringUTF8(s, offset, length)

Так же, как substring, но для кодовых точек Unicode. Работает при допущении, что строка содержит набор байтов, представляющий текст в кодировке UTF-8. Если допущение не выполнено, то возвращает какой-нибудь результат (не кидает исключение).

## appendTrailingCharIfAbsent(s, c)

Если строка s непустая и не содержит символ c на конце, то добавляет символ c в конец.

## convertCharset(s, from, to)

Возвращает сконвертированную из кодировки from в кодировку to строку s.

## base64Encode(s)

Производит кодирование строки s в base64-представление.

Синоним: TO\_BASE64.

## base64Decode(s)

Декодирует base64-представление s в исходную строку. При невозможности декодирования выбрасывает исключение

Синоним: FROM\_BASE64.

## tryBase64Decode(s)

Функционал аналогичен base64Decode, но при невозможности декодирования возвращает пустую строку.

## endsWith(s, suffix)

Возвращает 1, если строка завершается указанным суффиксом, и 0 в противном случае.

## startsWith(str, prefix)

Возвращает 1, если строка начинается указанным префиксом, в противном случае 0.

```
SELECT startsWith('Spider-Man', 'Spi');
```

### Возвращаемые значения

- 1, если строка начинается указанным префиксом.
- 0, если строка не начинается указанным префиксом.

### Пример

Запрос:

```
SELECT startsWith('Hello, world!', 'He');
```

Результат:

```
startsWith('Hello, world!', 'He')  
1 |
```

## trim

Удаляет все указанные символы с начала или окончания строки.

По умолчанию удаляет все последовательные вхождения обычных пробелов (32 символ ASCII) с обоих концов строки.

### Синтаксис

```
trim([[LEADING|TRAILING|BOTH] trim_character FROM] input_string)
```

### Аргументы

- `trim_character` — один или несколько символов, подлежащие удалению. [String](#).
- `input_string` — строка для обрезки. [String](#).

### Возвращаемое значение

Исходную строку после обрезки с левого и (или) правого концов строки.

Тип: [String](#).

### Пример

Запрос:

```
SELECT trim(BOTH '()' FROM '( Hello, world! )');
```

Результат:

```
trim(BOTH '()' FROM '( Hello, world! )')  
Hello, world! |
```

## trimLeft

Удаляет все последовательные вхождения обычных пробелов (32 символ ASCII) с левого конца строки. Не удаляет другие виды пробелов (табуляция, пробел без разрыва и т. д.).

### Синтаксис

```
trimLeft(input_string)
```

Алиас: `Itrim(input_string)`.

### Аргументы

- `input_string` — строка для обрезки. [String](#).

### Возвращаемое значение

Исходную строку без общих пробельных символов слева.

Тип: `String`.

## Пример

Запрос:

```
SELECT trimLeft('Hello, world!');
```

Результат:

```
trimLeft('Hello, world!')  
Hello, world!
```

## trimRight

Удаляет все последовательные вхождения обычных пробелов (32 символ ASCII) с правого конца строки. Не удаляет другие виды пробелов (табуляция, пробел без разрыва и т. д.).

### Синтаксис

```
trimRight(input_string)
```

Алиас: `rtrim(input_string)`.

### Аргументы

- `input_string` — строка для обрезки. `String`.

### Возвращаемое значение

Исходную строку без общих пробельных символов справа.

Тип: `String`.

## Пример

Запрос:

```
SELECT trimRight('Hello, world!');
```

Результат:

```
trimRight('Hello, world!')  
Hello, world!
```

## trimBoth

Удаляет все последовательные вхождения обычных пробелов (32 символ ASCII) с обоих концов строки. Не удаляет другие виды пробелов (табуляция, пробел без разрыва и т. д.).

### Синтаксис

```
trimBoth(input_string)
```

Алиас: `trim(input_string)`.

## Аргументы

- `input_string` — строка для обрезки. [String](#).

## Возвращаемое значение

Исходную строку без общих пробельных символов с обоих концов строки.

Тип: [String](#).

## Пример

Запрос:

```
SELECT trimBoth('Hello, world!');
```

Результат:

```
trimBoth('Hello, world!' )  
Hello, world!
```

## CRC32(s)

Возвращает чексумму CRC32 данной строки, используется CRC-32-IEEE 802.3 многочлен и начальным значением `0xffffffff` (т.к. используется реализация из zlib).

Тип результата — [UInt32](#).

## CRC32IEEE(s)

Возвращает чексумму CRC32 данной строки, используется CRC-32-IEEE 802.3 многочлен.

Тип результата — [UInt32](#).

## CRC64(s)

Возвращает чексумму CRC64 данной строки, используется CRC-64-ECMA многочлен.

Тип результата — [UInt64](#).

## normalizeQuery

Заменяет литералы, последовательности литералов и сложные псевдонимы заполнителями.

## Синтаксис

```
normalizeQuery(x)
```

## Аргументы

- `x` — последовательность символов. [String](#).

## Возвращаемое значение

- Последовательность символов с заполнителями.

Тип: [String](#).

## Пример

Запрос:

```
SELECT normalizeQuery('[1, 2, 3, x]') AS query;
```

Результат:

```
query
[?.., x] |
```

## normalizedQueryHash

Возвращает идентичные 64-битные хэш - суммы без значений литералов для аналогичных запросов.  
Это помогает анализировать журнал запросов.

## Синтаксис

```
normalizedQueryHash(x)
```

## Аргументы

- x — последовательность символов. [String](#).

## Возвращаемое значение

- Хэш-сумма.

Тип: [UInt64](#).

## Пример

Запрос:

```
SELECT normalizedQueryHash('SELECT 1 AS `xyz`') != normalizedQueryHash('SELECT 1 AS `abc`') AS res;
```

Результат:

```
res
1 |
```

## normalizeUTF8NFC

Преобразует строку в нормализованную форму [NFC](#), предполагая, что строка содержит набор байтов, составляющих текст в кодировке UTF-8.

## Синтаксис

```
normalizeUTF8NFC(words)
```

## Аргументы

- words — входная строка, которая содержит набор байтов, составляющих текст в кодировке UTF-8. [String](#).

## Возвращаемое значение

- Строка, преобразованная в нормализованную форму NFC.

Тип: [String](#).

## Пример

Запрос:

```
SELECT length('â'), normalizeUTF8NFC('â') AS nfc, length(nfc) AS nfc_len;
```

Результат:

length('â')	nfc	nfc_len
2   â	2	

## normalizeUTF8NFD

Преобразует строку в нормализованную форму [NFD](#), предполагая, что строка содержит набор байтов, составляющих текст в кодировке UTF-8.

## Синтаксис

```
normalizeUTF8NFD(words)
```

## Аргументы

- words — входная строка, которая содержит набор байтов, составляющих текст в кодировке UTF-8. [String](#).

## Возвращаемое значение

- Строка, преобразованная в нормализованную форму NFD.

Тип: [String](#).

## Пример

Запрос:

```
SELECT length('â'), normalizeUTF8NFD('â') AS nfd, length(nfd) AS nfd_len;
```

Результат:

length('â')	nfd	nfd_len
2   â	3	

## normalizeUTF8NFKC

Преобразует строку в нормализованную форму **NFKC**, предполагая, что строка содержит набор байтов, составляющих текст в кодировке UTF-8.

## Синтаксис

```
normalizeUTF8NFKC(words)
```

## Аргументы

- words — входная строка, которая содержит набор байтов, составляющих текст в кодировке UTF-8. **String**.

## Возвращаемое значение

- Строка, преобразованная в нормализованную форму NFKC.

Тип: **String**.

## Пример

Запрос:

```
SELECT length('â'), normalizeUTF8NFKC('â') AS nfkc, length(nfkc) AS nfkc_len;
```

Результат:

length('â')	nfkcs	nfkcs_len
2	â	2

## normalizeUTF8NFKD

Преобразует строку в нормализованную форму **NFKD**, предполагая, что строка содержит набор байтов, составляющих текст в кодировке UTF-8.

## Синтаксис

```
normalizeUTF8NFKD(words)
```

## Аргументы

- words — входная строка, которая содержит набор байтов, составляющих текст в кодировке UTF-8. **String**.

## Возвращаемое значение

- Строка, преобразованная в нормализованную форму NFKD.

Тип: **String**.

## Пример

Запрос:

```
SELECT length('â'), normalizeUTF8NFKD('â') AS nfkd, length(nfkd) AS nfkd_len;
```

Результат:



## encodeXMLComponent

Экранирует символы для размещения строки в текстовом узле или атрибуте XML.

Экранируются символы, которые в формате XML являются зарезервированными (служебными): <, &, >, ", '.

### Синтаксис

```
encodeXMLComponent(x)
```

### Аргументы

- `x` — последовательность символов. [String](#).

### Возвращаемое значение

- Стока, в которой зарезервированные символы экранированы.

Тип: [String](#).

### Пример

Запрос:

```
SELECT encodeXMLComponent('Hello, "world"!');  
SELECT encodeXMLComponent('<123>');  
SELECT encodeXMLComponent('&clickhouse');  
SELECT encodeXMLComponent('\foo');
```

Результат:

```
Hello, "world"!<123>&clickhouse'&apos;foo&apos;
```

## decodeXMLComponent

Заменяет символами предопределенные мнемоники XML: " & &apos; &gt; &lt;

Также эта функция заменяет числовые ссылки соответствующими символами юникод.

Поддерживаются десятичная (например, \10003;) и шестнадцатеричная (\x2713;) формы.

### Синтаксис

```
decodeXMLComponent(x)
```

### Аргументы

- `x` — последовательность символов. [String](#).

### Возвращаемое значение

- Стока с произведенными заменами.

Тип: [String](#).

## Пример

Запрос:

```
SELECT decodeXMLComponent('&apos;foo&apos;');
SELECT decodeXMLComponent('&lt; &#x3A3; &gt;');
```

Результат:

```
'foo'
< Σ >
```

## Смотрите также

- [Мнемоники в HTML](#)

## extractTextFromHTML

Функция для извлечения текста из HTML или XHTML.

Она не соответствует всем HTML, XML или XHTML стандартам на 100%, но ее реализация достаточно точная и быстрая. Правила обработки следующие:

1. Комментарии удаляются. Пример: `<!-- test -->`. Комментарий должен оканчиваться символами `-->`. Вложенные комментарии недопустимы.  
Примечание: конструкции наподобие `<!-->` и `<!---->` не являются допустимыми комментариями в HTML, но они будут удалены согласно другим правилам.
2. Содержимое CDATA вставляется дословно. Примечание: формат CDATA специфичен для XML/XHTML. Но он обрабатывается всегда по принципу "наилучшего возможного результата".
3. Элементы `script` и `style` удаляются вместе со всем содержимым. Примечание: предполагается, что закрывающий тег не может появиться внутри содержимого. Например, в JS строковый литерал должен быть экранирован как `"<\script>"`.  
Примечание: комментарии и CDATA возможны внутри `script` или `style` - тогда закрывающие теги не ищутся внутри CDATA. Пример: `<script><![CDATA[</script>]]></script>`. Но они ищутся внутри комментариев. Иногда возникают сложные случаи: `<script>var x = "<!--"; </script> var y = "-->"; alert(x + y);</script>`  
Примечание: `script` и `style` могут быть названиями пространств имен XML - тогда они не обрабатываются как обычные элементы `script` или `style`. Пример: `<script:a>Hello</script:a>`.  
Примечание: пробелы возможны после имени закрывающего тега: `</script >`, но не перед ним: `</ script>`.
4. Другие теги или элементы, подобные тегам, удаляются, а их внутреннее содержимое остается.  
Пример: `<a>.</a>`  
Примечание: ожидается, что такой HTML является недопустимым: `<a test="">></a>`  
Примечание: функция также удаляет подобные тегам элементы: `<>`, `<!---->`, и т. д.  
Примечание: если встречается тег без завершающего символа `>`, то удаляется этот тег и весь следующий за ним текст: `<hello`
5. Мнемоники HTML и XML не декодируются. Они должны быть обработаны отдельной функцией.

6. Пробелы в тексте удаляются и добавляются по следующим правилам:
- Пробелы в начале и в конце извлеченного текста удаляются.
  - Несколько пробелов подряд заменяются одним пробелом.
  - Если текст разделен другими удаляемыми элементами и в этом месте нет пробела, он добавляется.
  - Это может привести к появлению неестественного написания, например: Hello**world**, Hello<-->world — в HTML нет пробелов, но функция вставляет их. Также следует учитывать такие варианты написания: Hello

world

, Hello  
world. Подобные результаты выполнения функции могут использоваться для анализа данных, например, для преобразования HTML-текста в набор используемых слов.
7. Также обратите внимание, что правильная обработка пробелов требует поддержки `<pre>` и свойств CSS `display` и `white-space`.

## Синтаксис

```
extractTextFromHTML(x)
```

### Аргументы

- `x` — текст для обработки. **String**.

### Возвращаемое значение

- Извлеченный текст.

Тип: **String**.

### Пример

Первый пример содержит несколько тегов и комментарий. На этом примере также видно, как обрабатываются пробелы.

Второй пример показывает обработку CDATA и тега script.

В третьем примере текст выделяется из полного HTML ответа, полученного с помощью функции **url**.

Запрос:

```
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.donothngfor2minutes.com/', RawBLOB, 'html String');
```

Результат:

```
A text with tags .
The content within <b>CDATA</b>
Do Nothing for 2 Minutes 2:00 &nbsp;
```

## ФУНКЦИИ ПОИСКА В СТРОКАХ

Во всех функциях, поиск регистрозависимый по умолчанию. Существуют варианты функций для регистрационезависимого поиска.

### position(haystack, needle), locate(haystack, needle)

Поиск подстроки `needle` в строке `haystack`.

Возвращает позицию (в байтах) найденной подстроки в строке, начиная с 1, или 0, если подстрока не найдена.

Для поиска без учета регистра используйте функцию [positionCaseInsensitive](#).

## Синтаксис

```
position(haystack, needle[, start_pos])
```

```
position(needle IN haystack)
```

Алиас: `locate(haystack, needle[, start_pos])`.

## Примечание

Синтаксис `position(needle IN haystack)` обеспечивает совместимость с SQL, функция работает также, как `position(haystack, needle)`.

## Аргументы

- `haystack` — строка, по которой выполняется поиск. [Строка](#).
- `needle` — подстрока, которую необходимо найти. [Строка](#).
- `start_pos` — опциональный параметр, позиция символа в строке, с которого начинается поиск. [UInt](#).

## Возвращаемые значения

- Начальная позиция в байтах (начиная с 1), если подстрока найдена.
- 0, если подстрока не найдена.

Тип: `Integer`.

## Примеры

Фраза «Hello, world!» содержит набор байт, представляющий текст в однобайтовой кодировке. Функция возвращает ожидаемый результат:

Запрос:

```
SELECT position('Hello, world!', '!');
```

Результат:

```
position('Hello, world!', '!')  
13 |
```

Аналогичная фраза на русском содержит символы, которые не могут быть представлены в однобайтовой кодировке. Функция возвращает неожиданный результат (используйте функцию [positionUTF8](#) для символов, которые не могут быть представлены одним байтом):

Запрос:

```
SELECT position('Привет, мир!', '!');
```

Результат:

```
position('Привет, мир!', '!')  
21 |
```

### Примеры работы функции с синтаксисом POSITION(needle IN haystack)

Запрос:

```
SELECT 1 = position('абв' IN 'абв');
```

Результат:

```
equals(1, position('абв', 'абв'))  
1 |
```

Запрос:

```
SELECT 0 = position('абв' IN "");
```

Результат:

```
equals(0, position("", 'абв'))  
1 |
```

## positionCaseInsensitive

Такая же, как и [position](#), но работает без учета регистра. Возвращает позицию в байтах найденной подстроки в строке, начиная с 1.

Работает при допущении, что строка содержит набор байт, представляющий текст в однобайтовой кодировке. Если допущение не выполнено — то возвращает неопределенный результат (не кидает исключение). Если символ может быть представлен с помощью двух байтов, он будет представлен двумя байтами и так далее.

### Синтаксис

```
positionCaseInsensitive(haystack, needle[, start_pos])
```

### Аргументы

- `haystack` — строка, по которой выполняется поиск. [Строка](#).
- `needle` — подстрока, которую необходимо найти. [Строка](#).
- `start_pos` — опциональный параметр, позиция символа в строке, с которого начинается поиск. [UInt](#).

### Возвращаемые значения

- Начальная позиция в байтах (начиная с 1), если подстрока найдена.

- 0, если подстрока не найдена.

Тип: Integer.

## Пример

Запрос:

```
SELECT positionCaseInsensitive('Hello, world!', 'hello');
```

Результат:

```
positionCaseInsensitive('Hello, world!', 'hello')  
1 |
```

## positionUTF8

Возвращает позицию (в кодовых точках Unicode) найденной подстроки в строке, начиная с 1.

Работает при допущении, что строка содержит набор кодовых точек, представляющий текст в кодировке UTF-8. Если допущение не выполнено — то возвращает неопределенный результат (не кидает исключение). Если символ может быть представлен с помощью двух кодовых точек, он будет представлен двумя и так далее.

Для поиска без учета регистра используйте функцию [positionCaseInsensitiveUTF8](#).

## Синтаксис

```
positionUTF8(haystack, needle[, start_pos])
```

## Аргументы

- `haystack` — строка, по которой выполняется поиск. [Строка](#).
- `needle` — подстрока, которую необходимо найти. [Строка](#).
- `start_pos` — опциональный параметр, позиция символа в строке, с которого начинается поиск. [UInt](#).

## Возвращаемые значения

- Начальная позиция в кодовых точках Unicode (начиная с 1), если подстрока найдена.
- 0, если подстрока не найдена.

Тип: Integer.

## Примеры

Фраза «Привет, мир!» содержит набор символов, каждый из которых можно представить с помощью одной кодовой точки. Функция возвращает ожидаемый результат:

Запрос:

```
SELECT positionUTF8('Привет, мир!', '!');
```

Результат:

```
positionUTF8('Привет, мир!', '!')  
12 |
```

Фраза «Salut, étudiante!» содержит символ é, который может быть представлен одной кодовой точкой (U+00E9) или двумя (U+0065U+0301). Поэтому функция positionUTF8() может вернуть неожиданный результат:

Запрос для символа é, который представлен одной кодовой точкой U+00E9:

```
SELECT positionUTF8('Salut, étudiante!', '!');
```

Result:

```
positionUTF8('Salut, étudiante!', '!')  
17 |
```

Запрос для символа é, который представлен двумя кодовыми точками U+0065U+0301:

```
SELECT positionUTF8('Salut, étudiante!', '!');
```

Результат:

```
positionUTF8('Salut, étudiante!', '!')  
18 |
```

## positionCaseInsensitiveUTF8

Такая же, как и [positionUTF8](#), но работает без учета регистра. Возвращает позицию (в кодовых точках Unicode) найденной подстроки в строке, начиная с 1.

Работает при допущении, что строка содержит набор кодовых точек, представляющий текст в кодировке UTF-8. Если допущение не выполнено — то возвращает неопределенный результат (не кидает исключение). Если символ может быть представлен с помощью двух кодовых точек, он будет представлен двумя и так далее.

### Синтаксис

```
positionCaseInsensitiveUTF8(haystack, needle[, start_pos])
```

### Аргументы

- `haystack` — строка, по которой выполняется поиск. [Строка](#).
- `needle` — подстрока, которую необходимо найти. [Строка](#).
- `start_pos` — опциональный параметр, позиция символа в строке, с которого начинается поиск. [UInt](#).

### Возвращаемые значения

- Начальная позиция в байтах (начиная с 1), если подстрока найдена.
- 0, если подстрока не найдена.

Тип: Integer.

## Пример

Запрос:

```
SELECT positionCaseInsensitiveUTF8('Привет, мир!', 'Мир');
```

Результат:

```
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])
```

## Parameters

- `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

Смотрите [multiSearchAllPositions](#).

## `multiSearchFirstPosition(haystack, [needle1, needle2, ..., needlen])`

Так же, как и `position`, только возвращает оффсет первого вхождения любого из `needles`.

Для поиска без учета регистра и/или в кодировке UTF-8 используйте функции `multiSearchFirstPositionCaseInsensitive`, `multiSearchFirstPositionUTF8`, `multiSearchFirstPositionCaseInsensitiveUTF8`.

## `multiSearchFirstIndex(haystack, [needle1, needle2, ..., needlen])`

Возвращает индекс  $i$  (нумерация с единицы) первой найденной строки  $needle_i$  в строке `haystack` и 0 иначе.

Для поиска без учета регистра и/или в кодировке UTF-8 используйте функции `multiSearchFirstIndexCaseInsensitive`, `multiSearchFirstIndexUTF8`, `multiSearchFirstIndexCaseInsensitiveUTF8`.

## `multiSearchAny(haystack, [needle1, needle2, ..., needlen])`

Возвращает 1, если хотя бы одна подстрока  $needle_i$  нашлась в строке `haystack` и 0 иначе.

Для поиска без учета регистра и/или в кодировке UTF-8 используйте функции `multiSearchAnyCaseInsensitive`, `multiSearchAnyUTF8`, `multiSearchAnyCaseInsensitiveUTF8`.

### Примечание

Во всех функциях `multiSearch*` количество `needles` должно быть меньше  $2^8$  из-за особенностей реализации.

## `match(haystack, pattern)`

Проверка строки на соответствие регулярному выражению `pattern`. Регулярное выражение **re2**. Синтаксис регулярных выражений **re2** является более ограниченным по сравнению с регулярными выражениями **Perl** ([подробнее](#)).

Возвращает 0 (если не соответствует) или 1 (если соответствует).

Обратите внимание, что для экранирования в регулярном выражении, используется символ `\` (обратный слеш). Этот же символ используется для экранирования в строковых литералах. Поэтому, чтобы экранировать символ в регулярном выражении, необходимо написать в строковом литерале `\` (два обратных слеша).

Регулярное выражение работает со строкой как с набором байт. Регулярное выражение не может содержать нулевые байты.

Для шаблонов на поиск подстроки в строке, лучше используйте `LIKE` или `position`, так как они работают существенно быстрее.

## `multiMatchAny(haystack, [pattern1, pattern2, ..., patternn])`

То же, что и `match`, но возвращает ноль, если ни одно регулярное выражение не подошло и один, если хотя бы одно. Используется библиотека **hyperscan** для соответствия регулярных выражений.

Для шаблонов на поиск многих подстрок в строке, лучше используйте `multiSearchAny`, так как она работает существенно быстрее.

### Примечание

Длина любой строки из `haystack` должна быть меньше  $2^{32}$  байт, иначе бросается исключение.  
Это ограничение связано с ограничением hyperscan API.

## `multiMatchAnyIndex(haystack, [pattern1, pattern2, ..., patternn])`

То же, что и `multiMatchAny`, только возвращает любой индекс подходящего регулярного выражения.

## `multiMatchAllIndices(haystack, [pattern1, pattern2, ..., patternn])`

То же, что и `multiMatchAny`, только возвращает массив всех индексов всех подходящих регулярных выражений в любом порядке.

## `multiFuzzyMatchAny(haystack, distance, [pattern1, pattern2, ..., patternn])`

То же, что и `multiMatchAny`, но возвращает 1 если любой pattern соответствует haystack в пределах константного **редакционного расстояния**. Эта функция также находится в экспериментальном режиме и может быть очень медленной. За подробностями обращайтесь к [документации hyperscan](#).

## `multiFuzzyMatchAnyIndex(haystack, distance, [pattern1, pattern2, ..., patternn])`

То же, что и `multiFuzzyMatchAny`, только возвращает любой индекс подходящего регулярного выражения в пределах константного редакционного расстояния.

## `multiFuzzyMatchAllIndices(haystack, distance, [pattern1, pattern2, ..., patternn])`

То же, что и `multiFuzzyMatchAny`, только возвращает массив всех индексов всех подходящих регулярных выражений в любом порядке в пределах константного редакционного расстояния.

### Примечание

`multiFuzzyMatch*` функции не поддерживают UTF-8 закодированные регулярные выражения, и такие выражения рассматриваются как байтовые из-за ограничения hyperscan.

### Примечание

Чтобы выключить все функции, использующие hyperscan, используйте настройку `SET allow_hyperscan = 0;`

## `extract(haystack, pattern)`

Извлечение фрагмента строки по регулярному выражению. Если haystack не соответствует регулярному выражению pattern, то возвращается пустая строка. Если регулярное выражение не содержит subpattern-ов, то вынимается фрагмент, который подпадает под всё регулярное выражение. Иначе вынимается фрагмент, который подпадает под первый subpattern.

## extractAll(haystack, pattern)

Извлечение всех фрагментов строки по регулярному выражению. Если haystack не соответствует регулярному выражению pattern, то возвращается пустая строка. Возвращается массив строк, состоящий из всех соответствий регулярному выражению. В остальном, поведение аналогично функции extract (по прежнему, вынимается первый subpattern, или всё выражение, если subpattern-а нет).

## extractAllGroupsHorizontal

Разбирает строку haystack на фрагменты, соответствующие группам регулярного выражения pattern. Возвращает массив массивов, где первый массив содержит все фрагменты, соответствующие первой группе регулярного выражения, второй массив - соответствующие второй группе, и т.д.

### Замечание

Функция extractAllGroupsHorizontal работает медленнее, чем функция [extractAllGroupsVertical](#).

### Синтаксис

```
extractAllGroupsHorizontal(haystack, pattern)
```

### Аргументы

- haystack — строка для разбора. Тип: [String](#).
- pattern — регулярное выражение, построенное по синтаксическим правилам [re2](#). Выражение должно содержать группы, заключенные в круглые скобки. Если выражение не содержит групп, генерируется исключение. Тип: [String](#).

### Возвращаемое значение

- Тип: [Array](#).

Если в строке haystack нет групп, соответствующих регулярному выражению pattern, возвращается массив пустых массивов.

### Пример

Запрос:

```
SELECT extractAllGroupsHorizontal('abc=111, def=222, ghi=333', '("[^"]+"|\w+)=(["^"]+"|\w+')';
```

Результат:

```
extractAllGroupsHorizontal('abc=111, def=222, ghi=333', '("[^"]+"|\w+)=(["^"]+"|\w+')')  
[['abc','def','ghi'],['111','222','333']]
```

### Смотрите также

- Функция [extractAllGroupsVertical](#)

## extractAllGroupsVertical

Разбирает строку haystack на фрагменты, соответствующие группам регулярного выражения pattern. Возвращает массив массивов, где каждый массив содержит по одному фрагменту, соответствующему каждой группе регулярного выражения. Фрагменты группируются в массивы в соответствии с порядком появления в исходной строке.

## Синтаксис

```
extractAllGroupsVertical(haystack, pattern)
```

## Аргументы

- haystack — строка для разбора. Тип: [String](#).
- pattern — регулярное выражение, построенное по синтаксическим правилам [re2](#). Выражение должно содержать группы, заключенные в круглые скобки. Если выражение не содержит групп, генерируется исключение. Тип: [String](#).

## Возвращаемое значение

- Тип: [Array](#).

Если в строке haystack нет групп, соответствующих регулярному выражению pattern, возвращается пустой массив.

## Пример

Запрос:

```
SELECT extractAllGroupsVertical('abc=111, def=222, ghi=333', '("[^"]+"|\\"w+)=(["^"]+"|\\"w+')';
```

Результат:

```
extractAllGroupsVertical('abc=111, def=222, ghi=333', '("[^"]+"|\\"w+)=(["^"]+"|\\"w+')')  
[['abc','111'],['def','222'],['ghi','333']]
```

## Смотрите также

- Функция [extractAllGroupsHorizontal](#)

## like(haystack, pattern), оператор haystack LIKE pattern

Проверка строки на соответствие простому регулярному выражению.

Регулярное выражение может содержать метасимволы % и \_.

% обозначает любое количество любых байт (в том числе, нулевое количество символов).

\_ обозначает один любой байт.

Для экранирования метасимволов, используется символ \ (обратный слеш). Смотрите замечание об экранировании в описании функции match.

Для регулярных выражений вида %needle% действует более оптимальный код, который работает также быстро, как функция position.

Для остальных регулярных выражений, код аналогичен функции match.

## notLike(haystack, pattern), оператор haystack NOT LIKE pattern

То же, что like, но с отрицанием.

## ilike

Нечувствительный к регистру вариант функции `like`. Вы можете использовать оператор `ILIKE` вместо функции `ilike`.

### Синтаксис

```
ilike(haystack, pattern)
```

### Аргументы

- `haystack` — входная строка. `String`.
- `pattern` — если `pattern` не содержит процента или нижнего подчеркивания, тогда `pattern` представляет саму строку. Нижнее подчеркивание (`_`) в `pattern` обозначает любой отдельный символ. Знак процента (`%`) соответствует последовательности из любого количества символов: от нуля и более.

Некоторые примеры `pattern`:

```
'abc' ILIKE 'abc'    true
'abc' ILIKE 'a%'    true
'abc' ILIKE '_b_'   true
'abc' ILIKE 'c'     false
```

### Возвращаемые значения

- Правда, если строка соответствует `pattern`.
- Ложь, если строка не соответствует `pattern`.

### Пример

Входная таблица:

id	name	days
1	January	31
2	February	29
3	March	31
4	April	30

Запрос:

```
SELECT * FROM Months WHERE ilike(name, '%j%');
```

Результат:

id	name	days
1	January	31

### Смотрите также

[ngramDistance\(haystack, needle\)](#)

Вычисление 4-граммного расстояния между `haystack` и `needle`: считается симметрическая разность между двумя мульти множествами 4-грамм и нормализуется на сумму их мощностей. Возвращает число float от 0 до 1 – чем ближе к нулю, тем больше строки похожи друг на друга. Если константный `needle` или `haystack` больше чем 32КБ, кидается исключение. Если некоторые строки из неконстантного `haystack` или `needle` больше 32КБ, расстояние всегда равно единице.

Для поиска без учета регистра и/или в формате UTF-8 используйте функции `ngramDistanceCaseInsensitive`, `ngramDistanceUTF8`, `ngramDistanceCaseInsensitiveUTF8`.

## ngramSearch(haystack, needle)

То же, что и `ngramDistance`, но вычисляет несимметричную разность между `needle` и `haystack` – количество n-грамм из `needle` минус количество общих n-грамм, нормированное на количество n-грамм из `needle`. Чем ближе результат к единице, тем вероятнее, что `needle` внутри `haystack`. Может быть использовано для приближенного поиска.

Для поиска без учета регистра и/или в формате UTF-8 используйте функции `ngramSearchCaseInsensitive`, `ngramSearchUTF8`, `ngramSearchCaseInsensitiveUTF8`.

## Примечание

Для случая UTF-8 мы используем триграммное расстояние. Вычисление n-граммного расстояния не совсем честное. Мы используем 2-х байтные хэши для хэширования n-грамм, а затем вычисляем (не)симметрическую разность между хэш таблицами – могут возникнуть коллизии. В формате UTF-8 без учета регистра мы не используем честную функцию `tolower` – мы обнуляем 5-й бит (нумерация с нуля) каждого байта кодовой точки, а также первый бит нулевого байта, если байтов больше 1 – это работает для латиницы и почти для всех кириллических букв.

## countMatches(haystack, pattern)

Возвращает количество совпадений, найденных в строке `haystack`, для регулярного выражения `pattern`.

### Синтаксис

```
countMatches(haystack, pattern)
```

### Аргументы

- `haystack` — строка, по которой выполняется поиск. [String](#).
- `pattern` — регулярное выражение, построенное по синтаксическим правилам [re2](#). [String](#).

### Возвращаемое значение

- Количество совпадений.

Тип: [UInt64](#).

### Примеры

Запрос:

```
SELECT countMatches('foobar.com', 'o+');
```

Результат:

```
countMatches('foobar.com', 'o+')  
2 |
```

Запрос:

```
SELECT countMatches('aaaa', 'aa');
```

Результат:

```
countMatches('aaaa', 'aa')  
2 |
```

## countSubstrings

Возвращает количество вхождений подстроки.

Для поиска без учета регистра, используйте функции [countSubstringsCaseInsensitive](#) или [countSubstringsCaseInsensitiveUTF8](#)

### Синтаксис

```
countSubstrings(haystack, needle[, start_pos])
```

### Аргументы

- `haystack` — строка, в которой ведется поиск. [String](#).
- `needle` — искомая подстрока. [String](#).
- `start_pos` — позиция первого символа в строке, с которого начнется поиск. Необязательный параметр. [UInt](#).

### Возвращаемые значения

- Число вхождений.

Тип: [UInt64](#).

### Примеры

Запрос:

```
SELECT countSubstrings('foobar.com', '.');
```

Результат:

```
countSubstrings('foobar.com', '.')  
1 |
```

Запрос:

```
SELECT countSubstrings('aaaa', 'aa');
```

Результат:

```
countSubstrings('aaaa', 'aa')  
2 |
```

Запрос:

```
SELECT countSubstrings('abc__abc', 'abc', 4);
```

Результат:

```
countSubstrings('abc__abc', 'abc', 4)  
1 |
```

## countSubstringsCaseInsensitive

Возвращает количество вхождений подстроки без учета регистра.

### Синтаксис

```
countSubstringsCaseInsensitive(haystack, needle[, start_pos])
```

### Аргументы

- `haystack` — строка, в которой ведется поиск. [String](#).
- `needle` — искомая подстрока. [String](#).
- `start_pos` — позиция первого символа в строке, с которого начнется поиск. Необязательный параметр. [UInt](#).

### Возвращаемые значения

- Число вхождений.

Тип: [UInt64](#).

### Примеры

Запрос:

```
select countSubstringsCaseInsensitive('aba', 'B');
```

Результат:

```
countSubstringsCaseInsensitive('aba', 'B')  
1 |
```

Запрос:

```
SELECT countSubstringsCaseInsensitive('foobar.com', 'CoM');
```

Результат:

```
countSubstringsCaseInsensitive('foobar.com', 'CoM')—  
1 |
```

Запрос:

```
SELECT countSubstringsCaseInsensitive('abC__abC', 'aBc', 2);
```

Результат:

```
countSubstringsCaseInsensitive('abC__abC', 'aBc', 2)—  
1 |
```

## countSubstringsCaseInsensitiveUTF8

Возвращает количество вхождений подстроки в `UTF-8` без учета регистра.

### Синтаксис

```
SELECT countSubstringsCaseInsensitiveUTF8(haystack, needle[, start_pos])
```

### Аргументы

- `haystack` — строка, в которой ведется поиск. [String](#).
- `needle` — искомая подстрока. [String](#).
- `start_pos` — позиция первого символа в строке, с которого начнется поиск. Необязательный параметр. [UInt](#).

### Возвращаемые значения

- Число вхождений.

Тип: [UInt64](#).

### Примеры

Запрос:

```
SELECT countSubstringsCaseInsensitiveUTF8('абв', 'A');
```

Результат:

```
countSubstringsCaseInsensitiveUTF8('абв', 'A')—  
1 |
```

Запрос:

```
SELECT countSubstringsCaseInsensitiveUTF8('аБв_Абв_абв', 'Абв');
```

Результат:

```
countSubstringsCaseInsensitiveUTF8('аБв_Абв_абв', 'Абв')—  
3 |
```

## Функции поиска и замены в строках

### replaceOne(haystack, pattern, replacement)

Замена первого вхождения, если такое есть, подстроки pattern в haystack на подстроку replacement.

Здесь и далее, pattern и replacement должны быть константами.

### replaceAll(haystack, pattern, replacement)

Замена всех вхождений подстроки pattern в haystack на подстроку replacement.

### replaceRegexpOne(haystack, pattern, replacement)

Замена по регулярному выражению pattern. Регулярное выражение re2.

Заменяется только первое вхождение, если есть.

В качестве replacement может быть указан шаблон для замен. Этот шаблон может включать в себя подстановки \0-\9.

Подстановка \0 - вхождение регулярного выражения целиком. Подстановки \1-\9 - соответствующие по номеру subpattern-ы.

Для указания символа \ в шаблоне, он должен быть экранирован с помощью символа \\.

Также помните о том, что строковый литерал требует ещё одно экранирование.

Пример 1. Переведём дату в американский формат:

```
SELECT DISTINCT  
    EventDate,  
    replaceRegexpOne(toString(EventDate), '(\d{4})-(\d{2})-(\d{2})', '\\2/\\3/\\1') 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

Пример 2. Размножить строку десять раз:

```
SELECT replaceRegexpOne('Hello, World!', '.', '\0\0\0\0\0\0\0\0\0\0') AS res
```

```
res
| Hello, World!Hello, World!Hello, World!Hello, World!Hello, World!Hello, World!Hello,
World!Hello, World! |
```

## replaceRegexpAll(haystack, pattern, replacement)

То же самое, но делается замена всех вхождений. Пример:

```
SELECT replaceRegexpAll('Hello, World!', '.', '\\0\\0') AS res
```

```
res
HHeelllloo,, WWoorrlldd!! |
```

В качестве исключения, если регулярное выражение сработало на пустой подстроке, то замена делается не более одного раза.

Пример:

```
SELECT replaceRegexpAll('Hello, World!', '^', 'here: ') AS res
```

```
res
here: Hello, World! |
```

## Условные функции

### if

Условное выражение. В отличие от большинства систем, ClickHouse всегда считает оба выражения `then` и `else`.

#### Синтаксис

```
if(cond, then, else)
```

Если условие `cond` не равно нулю, то возвращается результат выражения `then`. Если условие `cond` равно нулю или является `NULL`, то результат выражения `then` пропускается и возвращается результат выражения `else`.

Чтобы вычислять функцию `if` по короткой схеме, используйте настройку `short_circuit_function_evaluation`. Если настройка включена, то выражение `then` вычисляется только для строк, где условие `cond` верно, а выражение `else` – для строк, где условие `cond` неверно.

Например, при выполнении запроса `SELECT if(number = 0, 0, intDiv(42, number)) FROM numbers(10)` не будет сгенерировано исключение из-за деления на ноль, так как `intDiv(42, number)` будет вычислено только для чисел, которые не удовлетворяют условию `number = 0`.

#### Аргументы

- `cond` – проверяемое условие. Может быть `UInt8` или `NULL`.

- `then` – возвращается результат выражения, если условие `cond` истинно.
- `else` – возвращается результат выражения, если условие `cond` ложно.

## Возвращаемые значения

Функция выполняет выражения `then` или `else` и возвращает его результат, в зависимости от того, было ли условие `cond` равно нулю или нет.

### Пример

Запрос:

```
SELECT if(1, plus(2, 2), plus(2, 6));
```

Результат:

```
+-----+  
| plus(2, 2) |  
+-----+  
     4 |
```

Запрос:

```
SELECT if(0, plus(2, 2), plus(2, 6));
```

Результат:

```
+-----+  
| plus(2, 6) |  
+-----+  
     8 |
```

## Тернарный оператор

Работает так же, как функция `if`.

Синтаксис: `cond ? then : else`

Возвращает `then`, если `cond` верно (больше нуля), в остальных случаях возвращает `else`.

- `cond` должно быть типа `UInt8`, `then` и `else` должны относиться к наименьшему общему типу.
- `then` и `else` могут быть `NULL`.

### Смотрите также

- [ifNotFinite](#).

## multilf

Позволяет более компактно записать оператор `CASE` в запросе.

### Синтаксис

```
multilf(cond_1, then_1, cond_2, then_2, ..., else)
```

Чтобы вычислять функцию `multilf` по короткой схеме, используйте настройку `short_circuit_function_evaluation`. Если настройка включена, то выражение `then_i` вычисляется только для строк, где условие `((NOT cond_1) AND (NOT cond_2) AND ... AND (NOT cond_{i-1}) AND cond_i)` верно, `cond_i` вычисляется только для строк, где условие `((NOT cond_1) AND (NOT cond_2) AND ... AND (NOT cond_{i-1}))` верно. Например, при выполнении запроса `SELECT multilf(number = 2, intDiv(1, number), number = 5) FROM numbers(10)` не будет сгенерировано исключение из-за деления на ноль.

## Аргументы

- `cond_N` — условие, при выполнении которого функция вернёт `then_N`.
- `then_N` — результат функции при выполнении.
- `else` — результат функции, если ни одно из условий не выполнено.

Функция принимает  $2N+1$  параметров.

## Возвращаемые значения

Функция возвращает одно из значений `then_N` или `else`, в зависимости от условий `cond_N`.

## Пример

Рассмотрим таблицу

x	y
1	NULL
2	3

Выполним запрос `SELECT multilf(isNull(y), x, y < 3, y, NULL) FROM t_null`. Результат:

```
multilf(isNull(y), x, less(y, 3), y, NULL)─
      1 |
      NULL |
```

# ФУНКЦИИ ДЛЯ РАБОТЫ С ФАЙЛАМИ

## file

Читает файл как строку. Содержимое файла не разбирается (не парсится) и записывается в указанную колонку в виде единой строки.

## Синтаксис

```
file(path)
```

## Аргументы

- `path` — относительный путь до файла от `user_files_path`. Путь к файлу может включать следующие символы подстановки и шаблоны: `*`, `?`, `{abc,def}` и `{N..M}`, где `N, M` — числа, `'abc'`, `'def'` — строки.

## Примеры

Вставка данных из файлов `a.txt` и `b.txt` в таблицу в виде строк:

```
INSERT INTO table SELECT file('a.txt'), file('b.txt');
```

## Смотрите также

- [user\\_files\\_path](#)
- [file](#)

# Математические функции

Все функции возвращают число типа Float64. Точность результата близка к максимально возможной, но результат может не совпадать с наиболее близким к соответствующему вещественному числу машинно представимым числом.

## e()

Возвращает число типа Float64, близкое к числу е.

## pi()

Возвращает число типа Float64, близкое к числу π.

## exp(x)

Принимает числовой аргумент, возвращает число типа Float64, близкое к экспоненте от аргумента.

## log(x)

Принимает числовой аргумент, возвращает число типа Float64, близкое к натуральному логарифму от аргумента.

## exp2(x)

Принимает числовой аргумент, возвращает число типа Float64, близкое к 2 в степени x.

## log2(x)

Принимает числовой аргумент, возвращает число типа Float64, близкое к двоичному логарифму от аргумента.

## exp10(x)

Принимает числовой аргумент, возвращает число типа Float64, близкое к 10 в степени x.

## log10(x)

Принимает числовой аргумент, возвращает число типа Float64, близкое к десятичному логарифму от аргумента.

## sqrt(x)

Принимает числовой аргумент, возвращает число типа Float64, близкое к квадратному корню от аргумента.

## cbrt(x)

Принимает числовой аргумент, возвращает число типа Float64, близкое к кубическому корню от аргумента.

## erf(x)

Если  $x$  неотрицательно, то  $\text{erf}(x / \sigma\sqrt{2})$  - вероятность того, что случайная величина, имеющая нормальное распределение со среднеквадратичным отклонением  $\sigma$ , принимает значение, отстоящее от мат. ожидания больше чем на  $x$ .

Пример (правило трёх сигм):

```
SELECT erf(3 / sqrt(2));
```

```
erf(divide(3, sqrt(2)))—  
0.9973002039367398 |
```

## erfc(x)

Принимает числовой аргумент, возвращает число типа Float64, близкое к  $1 - \text{erf}(x)$ , но без потери точности для больших  $x$ .

## lgamma(x)

Логарифм от гамма функции.

## tgamma(x)

Гамма функция.

## sin(x)

Синус.

## cos(x)

Косинус.

## tan(x)

Тангенс.

## asin(x)

Арксинус.

## acos(x)

Арккосинус.

## atan(x)

Арктангенс.

## pow(x, y)

Принимает два числовых аргумента  $x$  и  $y$ . Возвращает число типа Float64, близкое к  $x$  в степени  $y$ .

# $\cosh(x)$

Гиперболический косинус.

## Синтаксис

```
cosh(x)
```

## Аргументы

- $x$  — угол в радианах. Значения из интервала:  $-\infty < x < +\infty$ . **Float64**.

## Возвращаемое значение

- Значения из интервала:  $1 \leq \cosh(x) < +\infty$ .

Тип: **Float64**.

## Пример

Запрос:

```
SELECT cosh(0);
```

Результат:

```
└── cosh(0) ──  
      1 |
```

# $\text{acosh}(x)$

Обратный гиперболический косинус.

## Синтаксис

```
acosh(x)
```

## Аргументы

- $x$  — гиперболический косинус угла. Значения из интервала:  $1 \leq x < +\infty$ . **Float64**.

## Возвращаемое значение

- Угол в радианах. Значения из интервала:  $0 \leq \text{acosh}(x) < +\infty$ .

Тип: **Float64**.

## Пример

Запрос:

```
SELECT acosh(1);
```

Результат:

```
acosh(1)
0 |
```

## Смотрите также

- [cosh\(x\)](#)

## sinh(x)

Гиперболический синус.

## Синтаксис

```
sinh(x)
```

## Аргументы

- $x$  — угол в радианах. Значения из интервала:  $-\infty < x < +\infty$ . [Float64](#).

## Возвращаемое значение

- Значения из интервала:  $-\infty < \sinh(x) < +\infty$ .

Тип: [Float64](#).

## Пример

Запрос:

```
SELECT sinh(0);
```

Результат:

```
sinh(0)
0 |
```

## asinh(x)

Обратный гиперболический синус.

## Синтаксис

```
asinh(x)
```

## Аргументы

- $x$  — гиперболический синус угла. Значения из интервала:  $-\infty < x < +\infty$ . [Float64](#).

## Возвращаемое значение

- Угол в радианах. Значения из интервала:  $-\infty < \operatorname{asinh}(x) < +\infty$ .

Тип: [Float64](#).

## Пример

Запрос:

```
SELECT asinh(0);
```

Результат:

```
asinh(0)---  
0 |
```

## Смотрите также

- [sinh\(x\)](#)

## atanh(x)

Обратный гиперболический тангенс.

### Синтаксис

```
atanh(x)
```

### Аргументы

- $x$  — гиперболический тангенс угла. Значения из интервала:  $-1 < x < 1$ . [Float64](#).

### Возвращаемое значение

- Угол в радианах. Значения из интервала:  $-\infty < \operatorname{atanh}(x) < +\infty$ .

Тип: [Float64](#).

### Пример

Запрос:

```
SELECT atanh(0);
```

Результат:

```
atanh(0)---  
0 |
```

## atan2(y, x)

Функция вычисляет угол в радианах между положительной осью  $x$  и линией, проведенной из начала координат в точку  $(x, y) \neq (0, 0)$ .

### Синтаксис

```
atan2(y, x)
```

### Аргументы

- $y$  — координата у точки, в которую проведена линия. [Float64](#).

- $x$  — координата  $x$  точки, в которую проведена линия. [Float64](#).

## Возвращаемое значение

- Угол  $\theta$  в радианах из интервала:  $-\pi < \theta \leq \pi$ .

Тип: [Float64](#).

## Пример

Запрос:

```
SELECT atan2(1, 1);
```

Результат:

```
atan2(1, 1)
0.7853981633974483 |
```

## hypot( $x$ , $y$ )

Вычисляет длину гипотенузы прямоугольного треугольника. При использовании этой [функции](#) не возникает проблем при возведении в квадрат очень больших или очень малых чисел.

## Синтаксис

```
hypot(x, y)
```

## Аргументы

- $x$  — первый катет прямоугольного треугольника. [Float64](#).
- $y$  — второй катет прямоугольного треугольника. [Float64](#).

## Возвращаемое значение

- Длина гипотенузы прямоугольного треугольника.

Тип: [Float64](#).

## Пример

Запрос:

```
SELECT hypot(1, 1);
```

Результат:

```
hypot(1, 1)
1.4142135623730951 |
```

## log1p( $x$ )

Вычисляет  $\log(1+x)$ . [Функция](#)  $\log1p(x)$  является более точной, чем функция  $\log(1+x)$  для малых значений  $x$ .

## Синтаксис

```
log1p(x)
```

## Аргументы

- $x$  — значения из интервала:  $-1 < x < +\infty$ . **Float64**.

## Возвращаемое значение

- Значения из интервала:  $-\infty < \log1p(x) < +\infty$ .

Тип: **Float64**.

## Пример

Запрос:

```
SELECT log1p(0);
```

Результат:

```
log1p(0)
0
```

## Смотрите также

- [log\(x\)](#)

## sign(x)

Возвращает знак действительного числа.

## Синтаксис

```
sign(x)
```

## Аргумент

- $x$  — Значения от  $-\infty$  до  $+\infty$ . Любой числовой тип, поддерживаемый ClickHouse.

## Возвращаемое значение

- -1 если  $x < 0$
- 0 если  $x = 0$
- 1 если  $x > 0$

## Примеры

Результат sign() для нуля:

```
SELECT sign(0);
```

Результат:

```
sign(0) ┌─┐  
      0 |
```

Результат sign() для положительного аргумента:

```
SELECT sign(1);
```

Результат:

```
sign(1) ┌─┐  
      1 |
```

Результат sign() для отрицательного аргумента:

```
SELECT sign(-1);
```

Результат:

```
sign(-1) ┌─┐  
      -1 |
```

## ФУНКЦИИ ОКРУГЛЕНИЯ

### **floor(x[, N])**

Возвращает наибольшее круглое число, которое меньше или равно, чем x.

Круглым называется число, кратное  $1 / 10^N$  или ближайшее к нему число соответствующего типа данных, если  $1 / 10^N$  не представимо точно.

N - целочисленная константа, не обязательный параметр. По умолчанию - ноль, что означает - округлять до целого числа.

N может быть отрицательным.

Примеры: `floor(123.45, 1) = 123.4`, `floor(123.45, -1) = 120`.

x - любой числовой тип. Результат - число того же типа.

Для целочисленных аргументов имеет смысл округление с отрицательным значением N (для неотрицательных N, функция ничего не делает).

В случае переполнения при округлении (например, `floor(-128, -1)`), возвращается implementation specific результат.

### **ceil(x[, N])**

Возвращает наименьшее круглое число, которое больше или равно, чем x.

В остальном, аналогично функции floor, см. выше.

### **round(x[, N])**

Округляет значение до указанного десятичного разряда.

Функция возвращает ближайшее значение указанного порядка. В случае, когда заданное число равноудалено от чисел необходимого порядка, для типов с плавающей точкой (Float32/64) функция возвращает то из них, которое имеет ближайшую чётную цифру (банковское округление), для типов с фиксированной точкой (Decimal) функция использует округление в большую по модулю сторону (математическое округление).

```
round(expression [, decimal_places])
```

## Аргументы

- `expression` — число для округления. Может быть любым выражением, возвращающим числовой тип данных.
- `decimal-places` — целое значение.
  - Если `decimal-places > 0`, то функция округляет значение справа от запятой.
  - Если `decimal-places < 0` то функция округляет значение слева от запятой.
  - Если `decimal-places = 0`, то функция округляет значение до целого. В этом случае аргумент можно опустить.

## Возвращаемое значение:

Округлённое значение того же типа, что и входящее.

## Примеры

### Пример использования с 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

### Пример использования с Decimal

```
SELECT cast(number / 2 AS Decimal(10,4)) AS x, round(x) FROM system.numbers LIMIT 3
```

x	round(CAST(divide(number, 2), 'Decimal(10, 4)'))
0.0000	0.0000
0.5000	1.0000
1.0000	1.0000

## Примеры округления

Округление до ближайшего числа.

```
round(3.2, 0) = 3  
round(4.1267, 2) = 4.13  
round(22,-1) = 20  
round(467,-2) = 500  
round(-467,-2) = -500
```

Банковское округление.

```
round(3.5) = 4  
round(4.5) = 4  
round(3.55, 1) = 3.6  
round(3.65, 1) = 3.6
```

## Смотрите также

- [roundBankers](#)

## roundBankers

Округляет число до указанного десятичного разряда.

- Если округляемое число равноудалено от соседних чисел, то используется банковское округление.

Банковское округление (англ. banker's rounding) — метод округления дробных чисел. Если округляемое число равноудалено от соседних чисел, то оно округляется до ближайшей чётной цифры заданного десятичного разряда. К примеру, 3,5 округляется до 4, а 2,5 до 2.

Этот метод округления, используемый по умолчанию для чисел с плавающей запятой, определён в стандарте [IEEE 754]([https://en.wikipedia.org/wiki/IEEE\\_754#Roundings\\_to\\_nearest](https://en.wikipedia.org/wiki/IEEE_754#Roundings_to_nearest)). Функция [round] (#rounding\_functions-round) также округляет числа с плавающей запятой по этому методу. Функция `roundBankers` округляет не только числа с плавающей запятой, но и целые числа методом банковского округления, например, `roundBankers(45, -1) = 40`.

- В других случаях функция округляет к ближайшему целому.

Банковское округление позволяет уменьшить влияние округления чисел на результат суммирования или вычитания этих чисел.

Пример суммирования чисел 1.5, 2.5, 3.5 и 4.5 с различным округлением:

- Без округления:  $1.5 + 2.5 + 3.5 + 4.5 = 12$ .
- Банковское округление:  $2 + 2 + 4 + 4 = 12$ .
- Округление до ближайшего целого:  $2 + 3 + 4 + 5 = 14$ .

## Синтаксис

```
roundBankers(expression [, decimal_places])
```

## Аргументы

- `expression` — число для округления. Может быть любым выражением, возвращающим числовой тип данных.
- `decimal-places` — десятичный разряд. Целое число.
  - `decimal-places > 0` — функция округляет значение выражения до ближайшего чётного числа на соответствующей позиции справа от запятой. Например, `roundBankers(3.55, 1) = 3.6`.
  - `decimal-places < 0` — функция округляет значение выражения до ближайшего чётного числа на соответствующей позиции слева от запятой. Например, `roundBankers(24.55, -1) = 20`.
  - `decimal-places = 0` — функция округляет значение до целого. В этом случае аргумент можно не передавать. Например, `roundBankers(2.5) = 2`.

## Возвращаемое значение

Округлённое значение по методу банковского округления.

## Пример использования

Запрос:

```
SELECT number / 2 AS x, roundBankers(x, 0) AS b fROM system.numbers limit 10
```

Результат:

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

## Примеры банковского округления

```
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) = 11,76
```

## Смотрите также

- [round](#)

## roundToExp2(num)

Принимает число. Если число меньше единицы - возвращает 0. Иначе округляет число вниз до ближайшей (целой неотрицательной) степени двух.

## roundDuration(num)

Принимает число. Если число меньше единицы - возвращает 0. Иначе округляет число вниз до чисел из набора: 1, 10, 30, 60, 120, 180, 240, 300, 600, 1200, 1800, 3600, 7200, 18000, 36000. Эта функция специфична для Яндекс.Метрики и предназначена для реализации отчёта по длительности визита.

## roundAge(num)

Принимает число. Если число меньше 18 - возвращает 0. Иначе округляет число вниз до чисел из набора: 18, 25, 35, 45, 55. Эта функция специфична для Яндекс.Метрики и предназначена для реализации отчёта по возрасту посетителей.

# Функции для работы с контейнерами тар тар

Преобразовывает пары `ключ:значение` в тип данных `Map(key, value)`.

## Синтаксис

```
map(key1, value1[, key2, value2, ...])
```

## Аргументы

- `key` — ключ. `String` или `Integer`.
- `value` — значение. `String`, `Integer` или `Array`.

## Возвращаемое значение

- Структура данных в виде пар `ключ:значение`.

Тип: `Map(key, value)`.

## Примеры

Запрос:

```
SELECT map('key1', number, 'key2', number * 2) FROM numbers(3);
```

Результат:

```
map('key1', number, 'key2', multiply(number, 2))  
{'key1':0,'key2':0}  
{'key1':1,'key2':2}  
{'key1':2,'key2':4}
```

Запрос:

```
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;
```

Результат:

```
arrayElement(a, 'key2')  
0  
2  
4
```

## Смотрите также

- тип данных `Map(key, value)`

## mapAdd

Собирает все ключи и суммирует соответствующие значения.

## Синтаксис

```
mapAdd(arg1, arg2 [, ...])
```

## Аргументы

Аргументами являются контейнеры **Map** или **кортежи** из двух **массивов**, где элементы в первом массиве представляют ключи, а второй массив содержит значения для каждого ключа. Все массивы ключей должны иметь один и тот же тип, а все массивы значений должны содержать элементы, которые можно приводить к одному типу (**Int64**, **UInt64** или **Float64**). Общий приведенный тип используется в качестве типа для результирующего массива.

## Возвращаемое значение

- В зависимости от типа аргументов возвращает один **Map** или **кортеж**, в котором первый массив содержит отсортированные ключи, а второй — значения.

## Пример

Запрос с кортежем:

```
SELECT mapAdd(([toUInt8(1), 2], [1, 1]), ([toUInt8(1), 2], [1, 1])) as res, toTypeName(res) as type;
```

Результат:

```
res-----type-----  
([1,2],[2,2]) | Tuple(Array(UInt8), Array(UInt64)) |
```

Запрос с контейнером **Map**:

```
SELECT mapAdd(map(1,1), map(1,1));
```

Результат:

```
mapAdd(map(1, 1), map(1, 1))---  
{1:2}-----|-----
```

## mapSubtract

Собирает все ключи и вычитает соответствующие значения.

### Синтаксис

```
mapSubtract(Tuple(Array, Array), Tuple(Array, Array) [, ...])
```

## Аргументы

Аргументами являются контейнеры **Map** или **кортежи** из двух **массивов**, где элементы в первом массиве представляют ключи, а второй массив содержит значения для каждого ключа. Все массивы ключей должны иметь один и тот же тип, а все массивы значений должны содержать элементы, которые можно приводить к одному типу (**Int64**, **UInt64** или **Float64**). Общий приведенный тип используется в качестве типа для результирующего массива.

## Возвращаемое значение

- В зависимости от аргумента возвращает один **Map** или **кортеж**, в котором первый массив содержит отсортированные ключи, а второй — значения.

## Пример

Запрос:

```
SELECT mapSubtract(([toInt8(1), 2], [toInt32(1), 1]), ([toInt8(1), 2], [toInt32(2), 1])) as res, toTypeName(res) as type;
```

Результат:

res	type
[1,2],[-1,0]	Tuple(Array(UInt8), Array(Int64))

Запрос с контейнером **Map**:

```
SELECT mapSubtract(map(1,1), map(1,1));
```

Результат:

mapSubtract(map(1, 1), map(1, 1))	{1:0}
-----------------------------------	-------

## mapPopulateSeries

Заполняет недостающие ключи в контейнере **Map** (пара массивов ключей и значений), где ключи являются целыми числами. Кроме того, он поддерживает указание максимального ключа, который используется для расширения массива ключей.

### Синтаксис

```
mapPopulateSeries(keys, values[, max])
mapPopulateSeries(map[, max])
```

Генерирует контейнер **Map**, где ключи - это серия чисел, от минимального до максимального ключа (или аргумент **max**, если он указан), взятых из массива **keys** с размером шага один, и соответствующие значения, взятые из массива **values**. Если значение не указано для ключа, то в результирующем контейнере используется значение по умолчанию.

Количество элементов в **keys** и **values** должно быть одинаковым для каждой строки.

### Аргументы

Аргументами являются контейнер **Map** или два **массива**, где первый массив представляет ключи, а второй массив содержит значения для каждого ключа.

Сопоставленные массивы:

- **keys** — массив ключей. **Array(Int)**.
- **values** — массив значений. **Array(Int)**.
- **max** — максимальное значение ключа. Необязательный параметр. **Int8**, **Int16**, **Int32**, **Int64**, **Int128**, **Int256**.

или

- `map` — контейнер `Map` с целочисленными ключами. `Map`.

### Возвращаемое значение

- В зависимости от аргумента возвращает контейнер `Map` или `кортеж` из двух `массивов`: ключи отсортированные по порядку и значения соответствующих ключей.

### Пример

Запрос с сопоставленными массивами:

```
SELECT mapPopulateSeries([1,2,4], [11,22,44], 5) AS res, toTypeName(res) AS type;
```

Результат:

res	type
[1,2,3,4,5],[11,22,0,44,0]	Tuple(Array(UInt8), Array(UInt8))

Запрос с контейнером `Map`:

```
SELECT mapPopulateSeries(map(1, 10, 5, 20), 6);
```

Результат:

mapPopulateSeries(map(1, 10, 5, 20), 6)
{1:10,2:0,3:0,4:0,5:20,6:0}

## mapContains

Определяет, содержит ли контейнер `map` ключ `key`.

### Синтаксис

```
mapContains(map, key)
```

### Аргументы

- `map` — контейнер `Map`. `Map`.
- `key` — ключ. Тип соответствует типу ключей параметра `map`.

### Возвращаемое значение

- 1 если `map` включает `key`, иначе 0.

Тип: `UInt8`.

### Пример

Запрос:

```
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;
```

Результат:

```
mapContains(a, 'name')
  1 |
  0 |
```

## mapKeys

Возвращает все ключи контейнера `map`.

Функцию можно оптимизировать, если включить настройку `optimize_functions_to_subcolumns`. При `optimize_functions_to_subcolumns = 1` функция читает только подстолбец `keys` вместо чтения и обработки данных всего столбца. Запрос `SELECT mapKeys(m) FROM table` преобразуется к запросу `SELECT m.keys FROM table`.

### Синтаксис

```
mapKeys(map)
```

### Аргументы

- `map` — контейнер `Map`. [Map](#).

### Возвращаемое значение

- Массив со всеми ключами контейнера `map`.

Тип: [Array](#).

### Пример

Запрос:

```
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;
```

Результат:

```
mapKeys(a)
  ['name','age'] |
  ['number','position'] |
```

## mapValues

Возвращает все значения контейнера `map`.

Функцию можно оптимизировать, если включить настройку `optimize_functions_to_subcolumns`. При `optimize_functions_to_subcolumns = 1` функция читает только подстолбец `values` вместо чтения и обработки данных всего столбца. Запрос `SELECT mapValues(m) FROM table` преобразуется к запросу `SELECT m.values FROM table`.

## Синтаксис

```
mapKeys(map)
```

## Аргументы

- `map` — контейнер Мар. [Map](#).

## Возвращаемое значение

- Массив со всеми значениями контейнера `map`.

Тип: [Array](#).

## Примеры

Запрос:

```
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;
```

Результат:

```
mapValues(a)  
[ 'eleven', '11' ]  
[ 'twelve', '6.0' ]
```

# ФУНКЦИИ разбиения и слияния строк и массивов

## splitByChar(separator, s)

Разбивает строку на подстроки, используя в качестве разделителя `separator`.  
`separator` должен быть константной строкой из ровно одного символа.

Возвращается массив выделенных подстрок. Могут выделяться пустые подстроки, если разделитель идёт в начале или в конце строки, или если идёт более одного разделителя подряд.

## Синтаксис

```
splitByChar(separator, s)
```

## Аргументы

- `separator` — разделитель, состоящий из одного символа. [String](#).
- `s` — разбиваемая строка. [String](#).

## Возвращаемые значения

Возвращает массив подстрок. Пустая подстрока, может быть возвращена, когда:

- Разделитель находится в начале или конце строки;
- Задано несколько последовательных разделителей;
- Исходная строка `s` пуста.

Тип: `Array(String)`.

## Пример

```
SELECT splitByChar(',', '1,2,3,abcde');
```

```
splitByChar(',', '1,2,3,abcde') └  
['1','2','3','abcde'] ┌
```

## splitByString(separator, s)

Разбивает строку на подстроки, разделенные строкой. В качестве разделителя использует константную строку `separator`, которая может состоять из нескольких символов. Если строка `separator` пуста, то функция разделит строку `s` на массив из символов.

## Синтаксис

```
splitByString(separator, s)
```

## Аргументы

- `separator` — разделитель. `String`.
- `s` — разбиваемая строка. `String`.

## Возвращаемые значения

Возвращает массив подстрок. Пустая подстрока, может быть возвращена, когда:

- Разделитель находится в начале или конце строки;
- Задано несколько последовательных разделителей;
- Исходная строка `s` пуста.

Тип: `Array(String)`.

## Примеры

```
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)

Разбивает строку на подстроки, разделенные регулярным выражением. В качестве разделителя используется строка регулярного выражения `regexp`. Если `regexp` пустая, функция разделит строку `s` на массив одиночных символов. Если для регулярного выражения совпадения не найдено, строка `s` не будет разбита.

### Синтаксис

```
splitByRegexp(regexp, s)
```

### Аргументы

- `regexp` — регулярное выражение. Константа [String](#) или [FixedString](#).
- `s` — разбиваемая строка. [String](#).

### Возвращаемые значения

Возвращает массив выбранных подстрок. Пустая подстрока может быть возвращена, если:

- Непустое совпадение с регулярным выражением происходит в начале или конце строки;
- Имеется несколько последовательных совпадений с непустым регулярным выражением;
- Исходная строка `s` пуста, а регулярное выражение не пустое.

Тип: [Array\(String\)](#).

### Примеры

Запрос:

```
SELECT splitByRegexp('\d+', 'a12bc23de345f');
```

Результат:

```
splitByRegexp('\d+', 'a12bc23de345f')  
['a','bc','de','f'] |
```

Запрос:

```
SELECT splitByRegexp("", 'abcde');
```

Результат:

```
splitByRegexp("", 'abcde')  
['a','b','c','d','e'] |
```

## splitByWhitespace(s)

Разбивает строку на подстроки, используя в качестве разделителей пробельные символы.

## Синтаксис

```
splitByWhitespace(s)
```

### Аргументы

- `s` — разбиваемая строка. [String](#).

### Возвращаемые значения

Возвращает массив подстрок.

Тип: [Array\(String\)](#).

### Пример

```
SELECT splitByWhitespace(' 1! a, b. ');
```

```
splitByWhitespace(' 1! a, b. ')—  
['1','a','b.'] |
```

## splitByNonAlpha(s)

Разбивает строку на подстроки, используя в качестве разделителей пробельные символы и символы пунктуации.

## Синтаксис

```
splitByNonAlpha(s)
```

### Аргументы

- `s` — разбиваемая строка. [String](#).

### Возвращаемые значения

Возвращает массив подстрок.

Тип: [Array\(String\)](#).

### Пример

```
SELECT splitByNonAlpha(' 1! a, b. ');
```

```
splitByNonAlpha(' 1! a, b. ')—  
['1','a','b'] |
```

## arrayStringConcat(arr[, separator])

Склейывает строки, перечисленные в массиве, с разделителем `separator`.  
`separator` - необязательный параметр, константная строка, по умолчанию равен пустой строке.  
Возвращается строка.

## alphaTokens(s)

Выделяет подстроки из подряд идущих байт из диапазонов a-z и A-Z.  
Возвращается массив выделенных подстрок.

### Пример:

```
SELECT alphaTokens('abca1abc');
```

```
alphaTokens('abca1abc')—  
['abca','abc'] |
```

## ngrams

Выделяет из UTF-8 строки отрезки (n-граммы) размером ngramsize символов.

### Синтаксис

```
ngrams(string, ngramsize)
```

### Аргументы

- `string` — строка. [String](#) or [FixedString](#).
- `ngramsize` — размер n-грамм. [UInt](#).

### Возвращаемые значения

- Массив с n-граммами.

Тип: [Array\(FixedString\)](#).

### Пример

Запрос:

```
SELECT ngrams('ClickHouse', 3);
```

Результат:

```
ngrams('ClickHouse', 3)—  
['Cli','lic','ick','ckH','kHo','Hou','ous','use'] |
```

## tokens

Разбивает строку на токены, используя в качестве разделителей не буквенно-цифровые символы ASCII.

### Аргументы

- `input_string` — набор байтов. [String](#).

### Возвращаемые значения

Возвращает массив токенов.

Тип: **Array**.

## Пример

Запрос:

```
SELECT tokens('test1,;\\ test2,;\\ test3,;\\  test4') AS tokens;
```

Результат:

```
tokens  
['test1','test2','test3','test4'] |
```

## Битовые функции

Битовые функции работают для любой пары типов из UInt8, UInt16, UInt32, UInt64, Int8, Int16, Int32, Int64, Float32, Float64.

Тип результата - целое число, битность которого равна максимальной битности аргументов. Если хотя бы один аргумент знаковый, то результат - знаковое число. Если аргумент - число с плавающей запятой - оно приводится к Int64.

**bitAnd(a, b)**

**bitOr(a, b)**

**bitXor(a, b)**

**bitNot(a)**

**bitShiftLeft(a, b)**

**bitShiftRight(a, b)**

**bitTest**

Принимает любое целое число и конвертирует его в [двоичное число](#), возвращает значение бита в указанной позиции. Отсчет начинается с 0 справа налево.

## Синтаксис

```
SELECT bitTest(number, index)
```

## Аргументы

- `number` – целое число.
- `index` – позиция бита.

## Возвращаемое значение

Возвращает значение бита в указанной позиции.

Тип: `UInt8`.

## Пример

Например, число 43 в двоичной системе счисления равно: 101011.

Запрос:

```
SELECT bitTest(43, 1);
```

Результат:

```
bitTest(43, 1)  
 1 |
```

Другой пример:

Запрос:

```
SELECT bitTest(43, 2);
```

Результат:

```
bitTest(43, 2)  
 0 |
```

## bitTestAll

Возвращает результат **логической конъюнкции** (оператор AND) всех битов в указанных позициях. Отсчет начинается с 0 справа налево.

Бинарная конъюнкция:

0 AND 0 = 0

0 AND 1 = 0

1 AND 0 = 0

1 AND 1 = 1

## Синтаксис

```
SELECT bitTestAll(number, index1, index2, index3, index4, ...)
```

## Аргументы

- `number` – целое число.
- `index1, index2, index3, index4` – позиция бита. Например, конъюнкция для набора позиций `index1, index2, index3, index4` является истинной, если все его позиции истинны `index1  $\wedge$  index2  $\wedge$  index3  $\wedge$  index4`.

## Возвращаемое значение

Возвращает результат логической конъюнкции.

Тип: `UInt8`.

## Пример

Например, число 43 в двоичной системе счисления равно: 101011.

Запрос:

```
SELECT bitTestAll(43, 0, 1, 3, 5);
```

Результат:

```
bitTestAll(43, 0, 1, 3, 5) └  
      1 |
```

Другой пример:

Запрос:

```
SELECT bitTestAll(43, 0, 1, 3, 5, 2);
```

Результат:

```
bitTestAll(43, 0, 1, 3, 5, 2) └  
      0 |
```

## bitTestAny

Возвращает результат [логической дизъюнкции](#) (оператор OR) всех битов в указанных позициях.  
Отсчет начинается с 0 справа налево.

Бинарная дизъюнкция:

0 OR 0 = 0  
0 OR 1 = 1  
1 OR 0 = 1  
1 OR 1 = 1

### Синтаксис

```
SELECT bitTestAny(number, index1, index2, index3, index4, ...)
```

### Аргументы

- `number` – целое число.
- `index1, index2, index3, index4` – позиции бита.

### Возвращаемое значение

Возвращает результат логической дизъюнкции.

Тип: UInt8.

### Пример

Например, число 43 в двоичной системе счисления равно: 101011.

Запрос:

```
SELECT bitTestAny(43, 0, 2);
```

Результат:

```
bitTestAny(43, 0, 2)─  
 1 |
```

Другой пример:

Запрос:

```
SELECT bitTestAny(43, 4, 2);
```

Результат:

```
bitTestAny(43, 4, 2)─  
 0 |
```

## bitCount

Подсчитывает количество равных единице бит в числе.

### Синтаксис

```
bitCount(x)
```

### Аргументы

- $x$  — целое число или число с плавающей запятой. Функция использует представление числа в памяти, что позволяет поддержать числа с плавающей запятой.

### Возвращаемое значение

- Количество равных единице бит во входном числе.

Функция не преобразует входное значение в более крупный тип (sign extension). Поэтому, например, `bitCount(toUInt8(-1)) = 8`.

Тип: UInt8.

### Пример

Возьмём к примеру число 333. Его бинарное представление — 0000000101001101.

Запрос:

```
SELECT bitCount(333);
```

Результат:

```
bitCount(100)─  
 5 |
```

# bitHammingDistance

Возвращает **расстояние Хэмминга** между битовыми представлениями двух целых чисел. Может быть использовано с функциями **SimHash** для проверки двух строк на схожесть. Чем меньше расстояние, тем больше вероятность, что строки совпадают.

## Синтаксис

```
bitHammingDistance(int1, int2)
```

## Аргументы

- `int1` — первое целое число. **Int64**.
- `int2` — второе целое число. **Int64**.

## Возвращаемое значение

- Расстояние Хэмминга.

Тип: **UInt8**.

## Примеры

Запрос:

```
SELECT bitHammingDistance(111, 121);
```

Результат:

```
bitHammingDistance(111, 121)─  
      3 |
```

Используя **SimHash**:

```
SELECT bitHammingDistance(ngramSimHash('cat ate rat'), ngramSimHash('rat ate cat'));
```

Результат:

```
bitHammingDistance(ngramSimHash('cat ate rat'), ngramSimHash('rat ate cat'))─  
      5 |
```

# ФУНКЦИИ ДЛЯ БИТОВЫХ МАСОК

## bitmapBuild

Создаёт битовый массив из массива целочисленных значений.

```
bitmapBuild(array)
```

## Аргументы

- `array` – массив типа **UInt\***.

## Пример

```
SELECT bitmapBuild([1, 2, 3, 4, 5]) AS res, toTypeName(res);
```

```
res---toTypeName(bitmapBuild([1, 2, 3, 4, 5]))-----  
| AggregateFunction(groupBitmap, UInt8) |
```

## bitmapToArray

Преобразует битовый массив в массив целочисленных значений.

```
bitmapToArray(bitmap)
```

### Аргументы

- bitmap – битовый массив.

## Пример

```
SELECT bitmapToArray(bitmapBuild([1, 2, 3, 4, 5])) AS res;
```

```
res  
[1,2,3,4,5] |
```

## bitmapSubsetLimit

Создает подмножество битмапа с n элементами, расположенными между range\_start и cardinality\_limit.

### Синтаксис

```
bitmapSubsetLimit(bitmap, range_start, cardinality_limit)
```

### Аргументы

- bitmap – битмап. [Bitmap object](#).
- range\_start – начальная точка подмножества. [UInt32](#).
- cardinality\_limit – верхний предел подмножества. [UInt32](#).

### Возвращаемое значение

Подмножество битмапа.

Тип: [Bitmap object](#).

## Пример

Запрос:

```
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;
```

Результат:

```
res  
[30,31,32,33,100,200,500] |
```

## subBitmap

Возвращает элементы битмапа, начиная с позиции `offset`. Число возвращаемых элементов ограничивается параметром `cardinality_limit`. Аналог строковой функции `substring`), но для битмапа.

### Синтаксис

```
subBitmap(bitmap, offset, cardinality_limit)
```

### Аргументы

- `bitmap` – битмап. Тип: [Bitmap object](#).
- `offset` – позиция первого элемента возвращаемого подмножества. Тип: [UInt32](#).
- `cardinality_limit` – максимальное число элементов возвращаемого подмножества. Тип: [UInt32](#).

### Возвращаемое значение

Подмножество битмапа.

Тип: [Bitmap object](#).

### Пример

Запрос:

```
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;
```

Результат:

```
res  
[10,11,12,13,14,15,16,17,18,19] |
```

## bitmapContains

Проверяет вхождение элемента в битовый массив.

```
bitmapContains(haystack, needle)
```

### Аргументы

- `haystack` – [объект Bitmap](#), в котором функция ищет значение.
- `needle` – значение, которое функция ищет. Тип — [UInt32](#).

### Возвращаемые значения

- 0 — если в `haystack` нет `needle`.

- 1 — если в `haystack` есть `needle`.

Тип — `UInt8`.

## Пример

```
SELECT bitmapContains(bitmapBuild([1,5,7,9]), toUInt32(9)) AS res;
```

```
res  
1 |
```

## bitmapHasAny

Проверяет, имеют ли два битовых массива хотя бы один общий элемент.

```
bitmapHasAny(bitmap1, bitmap2)
```

Если вы уверены, что `bitmap2` содержит строго один элемент, используйте функцию `bitmapContains`. Она работает эффективнее.

## Аргументы

- `bitmap*` – массив любого типа с набором элементов.

## Возвращаемые значения

- 1, если `bitmap1` и `bitmap2` имеют хотя бы один одинаковый элемент.
- 0, в противном случае.

## Пример

```
SELECT bitmapHasAny(bitmapBuild([1,2,3]),bitmapBuild([3,4,5])) AS res;
```

```
res  
1 |
```

## bitmapHasAll

Аналогично функции `hasAll(array, array)` возвращает 1 если первый битовый массив содержит все элементы второго, 0 в противном случае.

Если второй аргумент является пустым битовым массивом, то возвращает 1.

```
bitmapHasAll(bitmap,bitmap)
```

## Аргументы

- `bitmap` – битовый массив.

## Пример

```
SELECT bitmapHasAll(bitmapBuild([1,2,3]),bitmapBuild([3,4,5])) AS res;
```

```
res  
0 |
```

## bitmapAnd

Логическое И для двух битовых массивов. Результат — новый битовый массив.

```
bitmapAnd(bitmap,bitmap)
```

### Аргументы

- `bitmap` – битовый массив.

### Пример

```
SELECT bitmapToArray(bitmapAnd(bitmapBuild([1,2,3]),bitmapBuild([3,4,5]))) AS res;
```

```
res  
[3] |
```

## bitmapOr

Логическое ИЛИ для двух битовых массивов. Результат — новый битовый массив.

```
bitmapOr(bitmap,bitmap)
```

### Аргументы

- `bitmap` – битовый массив.

### Пример

```
SELECT bitmapToArray(bitmapOr(bitmapBuild([1,2,3]),bitmapBuild([3,4,5]))) AS res;
```

```
res  
[1,2,3,4,5] |
```

## bitmapXor

Логическое исключающее ИЛИ для двух битовых массивов. Результат — новый битовый массив.

```
bitmapXor(bitmap,bitmap)
```

### Аргументы

- `bitmap` – битовый массив.

### Пример

```
SELECT bitmapToArray(bitmapXor(bitmapBuild([1,2,3]),bitmapBuild([3,4,5]))) AS res;
```

```
res  
[1,2,4,5] |
```

## bitmapAndnot

Логическое отрицание И для двух битовых массивов. Результат — новый битовый массив.

```
bitmapAndnot(bitmap,bitmap)
```

### Аргументы

- bitmap – битовый массив.

### Пример

```
SELECT bitmapToArray(bitmapAndnot(bitmapBuild([1,2,3]),bitmapBuild([3,4,5]))) AS res;
```

```
res  
[1,2] |
```

## bitmapCardinality

Возвращает кардинальность битового массива в виде значения типа UInt64.

```
bitmapCardinality(bitmap)
```

### Аргументы

- bitmap – битовый массив.

### Пример

```
SELECT bitmapCardinality(bitmapBuild([1, 2, 3, 4, 5])) AS res;
```

```
res  
5 |
```

## bitmapAndCardinality

Выполняет логическое И и возвращает кардинальность (UInt64) результирующего битового массива.

```
bitmapAndCardinality(bitmap,bitmap)
```

### Аргументы

- bitmap – битовый массив.

## Пример

```
SELECT bitmapAndCardinality(bitmapBuild([1,2,3]),bitmapBuild([3,4,5])) AS res;
```

```
res  
1 |
```

## bitmapOrCardinality

Выполняет логическое ИЛИ и возвращает кардинальность (UInt64) результирующего битового массива.

```
bitmapOrCardinality(bitmap,bitmap)
```

## Аргументы

- bitmap – битовый массив.

## Пример

```
SELECT bitmapOrCardinality(bitmapBuild([1,2,3]),bitmapBuild([3,4,5])) AS res;
```

```
res  
5 |
```

## bitmapXorCardinality

Выполняет логическое исключающее ИЛИ и возвращает кардинальность (UInt64) результирующего битового массива.

```
bitmapXorCardinality(bitmap,bitmap)
```

## Аргументы

- bitmap – битовый массив.

## Пример

```
SELECT bitmapXorCardinality(bitmapBuild([1,2,3]),bitmapBuild([3,4,5])) AS res;
```

```
res  
4 |
```

## bitmapAndnotCardinality

Выполняет логическое отрицание И и возвращает кардинальность (UInt64) результирующего битового массива.

```
bitmapAndnotCardinality(bitmap,bitmap)
```

## Аргументы

- bitmap – битовый массив.

## Пример

```
SELECT bitmapAndNotCardinality(bitmapBuild([1,2,3]),bitmapBuild([3,4,5])) AS res;
```

res
2

# ФУНКЦИИ ХЭШИРОВАНИЯ

Функции хэширования могут использоваться для детерминированного псевдослучайного разбрасывания элементов.

Simhash – это хеш-функция, которая для близких значений возвращает близкий хеш.

## halfMD5

Интерпретирует все входные параметры как строки и вычисляет хеш MD5 для каждой из них. Затем объединяет хэши, берет первые 8 байт хэша результирующей строки и интерпретирует их как значение типа UInt64 с big-endian порядком байтов.

```
halfMD5(par1, ...)
```

Функция относительно медленная (5 миллионов коротких строк в секунду на ядро процессора). По возможности, используйте функцию [sipHash64](#) вместо неё.

## Аргументы

Функция принимает переменное число входных параметров. Аргументы могут быть любого поддерживаемого типа данных.

## Возвращаемое значение

Значение хэша с типом данных [UInt64](#).

## Пример

```
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

Вычисляет MD4 от строки и возвращает полученный набор байт в виде FixedString(16).

# MD5

Вычисляет MD5 от строки и возвращает полученный набор байт в виде FixedString(16). Если вам не нужен конкретно MD5, а нужен неплохой криптографический 128-битный хэш, то используйте вместо этого функцию sipHash128. Если вы хотите получить такой же результат, как выдаёт утилита md5sum, напишите lower(hex(MD5(s))).

## sipHash64

Генерирует 64-х битное значение SipHash.

```
sipHash64(par1,...)
```

Это криптографическая хэш-функция. Она работает по крайней мере в три раза быстрее, чем функция MD5.

Функция интерпретирует все входные параметры как строки и вычисляет хэш MD5 для каждой из них. Затем комбинирует хэши по следующему алгоритму.

1. После хэширования всех входных параметров функция получает массив хэшей.
2. Функция принимает первый и второй элементы и вычисляет хэш для массива из них.
3. Затем функция принимает хэш-значение, вычисленное на предыдущем шаге, и третий элемент исходного хэш-массива, и вычисляет хэш для массива из них.
4. Предыдущий шаг повторяется для всех остальных элементов исходного хэш-массива.

### Аргументы

Функция принимает переменное число входных параметров. Аргументы могут быть любого поддерживаемого типа данных.

### Возвращаемое значение

Значение хэша с типом данных UInt64.

### Пример

```
SELECT sipHash64(array('e','x','a'), 'mple', 10, toDateTime('2019-06-15 23:00:00')) AS SipHash, toTypeName(SipHash) AS type;
```

13726873534472839665	SipHash	type
		UInt64

## sipHash128

Вычисляет SipHash от строки.

Принимает аргумент типа String. Возвращает FixedString(16).

Отличается от sipHash64 тем, что финальный xor-folding состояния делается только до 128 бит.

## cityHash64

Генерирует 64-х битное значение CityHash.

```
cityHash64(par1,...)
```

Это не криптографическая хэш-функция. Она использует CityHash алгоритм для строковых параметров и зависящую от реализации быструю некриптографическую хэш-функцию для параметров с другими типами данных. Функция использует комбинатор CityHash для получения конечных результатов.

## Аргументы

Функция принимает переменное число входных параметров. Аргументы могут быть любого поддерживаемого типа данных.

## Возвращаемое значение

Значение хэша с типом данных **UInt64**.

## Примеры

Пример вызова:

```
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

А вот так вы можете вычислить чексумму всей таблицы с точностью до порядка строк:

```
SELECT groupBitXor(cityHash64(*)) FROM table
```

## intHash32

Вычисляет 32-битный хэш-код от целого числа любого типа.

Это сравнительно быстрая не криптографическая хэш-функция среднего качества для чисел.

## intHash64

Вычисляет 64-битный хэш-код от целого числа любого типа.

Работает быстрее, чем intHash32. Качество среднее.

## SHA1, SHA224, SHA256, SHA512

Вычисляет SHA-1, SHA-224, SHA-256, SHA-512 хеш строки и возвращает полученный набор байт в виде **FixedString**.

## Синтаксис

```
SHA1('s')  
...  
SHA512('s')
```

Функция работает достаточно медленно (SHA-1 — примерно 5 миллионов коротких строк в секунду на одном процессорном ядре, SHA-224 и SHA-256 — примерно 2.2 миллионов).

Рекомендуется использовать эти функции лишь в тех случаях, когда вам нужна конкретная хеш-функция и вы не можете её выбрать.

Даже в этих случаях рекомендуется применять функцию оффлайн — заранее вычисляя значения при вставке в таблицу, вместо того чтобы применять её при выполнении **SELECT**.

## Параметры

- `s` — входная строка для вычисления хеша SHA. [String](#).

## Возвращаемое значение

- Хеш SHA в виде шестнадцатеричной некодированной строки `FixedString`. SHA-1 хеш как `FixedString(20)`, SHA-224 как `FixedString(28)`, SHA-256 — `FixedString(32)`, SHA-512 — `FixedString(64)`.

Тип: [FixedString](#).

## Пример

Используйте функцию [hex](#) для представления результата в виде строки с шестнадцатеричной кодировкой.

Запрос:

```
SELECT hex(SHA1('abc'));
```

Результат:

```
hex(SHA1('abc'))—————  
A9993E364706816ABA3E25717850C26C9CD0D89D |
```

## URLHash(url[, N])

Быстрая не криптографическая хэш-функция неплохого качества для строки, полученной из URL путём некоторой нормализации.

`URLHash(s)` - вычислить хэш от строки без одного завершающего символа `/`, `?` или `#` на конце, если там такой есть.

`URLHash(s, N)` - вычислить хэш от строки до N-го уровня в иерархии URL, без одного завершающего символа `/`, `?` или `#` на конце, если там такой есть.

Уровни аналогичные `URLHierarchy`. Функция специфична для Яндекс.Метрики.

## farmFingerprint64

## farmHash64

Создает 64-битное значение [FarmHash](#), независимое от платформы (архитектуры сервера), что важно, если значения сохраняются или используются для разбиения данных на группы.

```
farmFingerprint64(par1, ...)  
farmHash64(par1, ...)
```

Эти функции используют методы `Fingerprint64` и `Hash64` из всех [доступных методов](#).

## Аргументы

Функция принимает переменное число входных параметров. Аргументы могут быть любого [поддерживаемого типа данных](#).

## Возвращаемое значение

Значение хэша с типом данных [UInt64](#).

## Пример

```
SELECT farmHash64(array('e','x','a'), 'mple', 10, toDateTime('2019-06-15 23:00:00')) AS FarmHash,  
toTypeName(FarmHash) AS type;
```

FarmHash	type
17790458267262532859	UInt64

## javaHash

Вычисляет **JavaHash** от строки. **JavaHash** не отличается ни скоростью, ни качеством, поэтому эту функцию следует считать устаревшей. Используйте эту функцию, если вам необходимо получить значение хэша по такому же алгоритму.

```
SELECT javaHash("")
```

### Возвращаемое значение

Хэш-значение типа Int32.

Тип: **javaHash**.

### Пример

Запрос:

```
SELECT javaHash('Hello, world!');
```

Результат:

javaHash('Hello, world!')	-1880044555
---------------------------	-------------

## javaHashUTF16LE

Вычисляет **JavaHash** от строки, при допущении, что строка представлена в кодировке UTF-16LE.

### Синтаксис

```
javaHashUTF16LE(stringUtf16le)
```

### Аргументы

- `stringUtf16le` — строка в UTF-16LE.

### Возвращаемое значение

Хэш-значение типа Int32.

Тип: **javaHash**.

### Пример

Верный запрос для строки кодированной в UTF-16LE.

Запрос:

```
SELECT javaHashUTF16LE(convertCharset('test', 'utf-8', 'utf-16le'));
```

Результат:

```
javaHashUTF16LE(convertCharset('test', 'utf-8', 'utf-16le'))—  
3556498 |
```

## hiveHash

Вычисляет HiveHash от строки.

```
SELECT hiveHash("")
```

HiveHash — это результат JavaHash с обнулённым битом знака числа. Функция используется в Apache Hive вплоть до версии 3.0.

### Возвращаемое значение

Хэш-значение типа Int32.

Тип: hiveHash.

### Пример

Запрос:

```
SELECT hiveHash('Hello, world!');
```

Результат:

```
hiveHash('Hello, world!')—  
267439093 |
```

## metroHash64

Генерирует 64-х битное значение MetroHash.

```
metroHash64(par1, ...)
```

### Аргументы

Функция принимает переменное число входных параметров. Аргументы могут быть любого поддерживаемого типа данных.

### Возвращаемое значение

Значение хэша с типом данных UInt64.

### Пример

```
SELECT metroHash64(array('e','x','a'), 'mple', 10, toDate('2019-06-15 23:00:00')) AS MetroHash,  
toTypeName(MetroHash) AS type;
```

MetroHash	type
14235658766382344533	UInt64

## jumpConsistentHash

Вычисляет JumpConsistentHash от значения типа UInt64.

Имеет два параметра: ключ типа UInt64 и количество бакетов. Возвращает значение типа Int32.

Дополнительные сведениясмотрите по ссылке: [JumpConsistentHash](#)

## murmurHash2\_32, murmurHash2\_64

Генерирует значение [MurmurHash2](#).

```
murmurHash2_32(par1, ...)  
murmurHash2_64(par1, ...)
```

### Аргументы

Обе функции принимают переменное число входных параметров. Аргументы могут быть любого [поддерживаемого типа данных](#).

### Возвращаемое значение

- Функция `murmurHash2_32` возвращает значение типа [UInt32](#).
- Функция `murmurHash2_64` возвращает значение типа [UInt64](#).

### Пример

```
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

Вычисляет 64-битное значение [MurmurHash2](#), используя те же hash seed, что и [gcc](#).

### Синтаксис

```
gccMurmurHash(par1, ...);
```

### Аргументы

- `par1, ...` — переменное число параметров. Каждый параметр может быть любого из [поддерживаемых типов данных](#).

### Возвращаемое значение

- Вычисленный хэш-код.

Тип: [UInt64](#).

### Примеры

Запрос:

```
SELECT
    gccMurmurHash(1, 2, 3) AS res1,
    gccMurmurHash('a', [1, 2, 3], 4, (4, ['foo', 'bar'], 1, (1, 2))) AS res2
```

Результат:

res1	res2
12384823029245979431	1188926775431157506

## murmurHash3\_32, murmurHash3\_64

Генерирует значение [MurmurHash3](#).

```
murmurHash3_32(par1, ...)
murmurHash3_64(par1, ...)
```

### Аргументы

Обе функции принимают переменное число входных параметров. Аргументы могут быть любого [поддерживаемого типа данных](#).

### Возвращаемое значение

- Функция `murmurHash3_32` возвращает значение типа [UInt32](#).
- Функция `murmurHash3_64` возвращает значение типа [UInt64](#).

### Пример

```
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

Генерирует значение [MurmurHash3](#).

```
murmurHash3_128( expr )
```

### Аргументы

- `expr` — [выражение](#), возвращающее значение типа [String](#).

### Возвращаемое значение

Хэш-значение типа [FixedString\(16\)](#).

### Пример

```
SELECT hex(murmurHash3_128('example_string')) AS MurmurHash3, toTypeName(MurmurHash3) AS type;
```

```
MurmurHash3——— type———  
368A1A311CB7342253354B548E7E7E71 | String |
```

## xxHash32, xxHash64

Вычисляет xxHash от строки. Предлагается в двух вариантах: 32 и 64 бита.

```
SELECT xxHash32("")  
OR  
SELECT xxHash64("")
```

### Возвращаемое значение

Хэш-значение типа `UInt32` или `UInt64`.

Тип: `xxHash`.

### Пример

Запрос:

```
SELECT xxHash32('Hello, world!');
```

Результат:

```
xxHash32('Hello, world!')———  
834093149 |—————
```

### Смотрите также

- [xxHash](#).

## ngramSimHash

Выделяет из ASCII строки отрезки (n-граммы) размером `ngramsize` символов и возвращает n-граммовый `simhash`. Функция регистрозависимая.

Может быть использована для проверки двух строк на схожесть вместе с функцией `bitHammingDistance`. Чем меньше [расстояние Хэмминга](#) между результатом вычисления `simhash` двух строк, тем больше вероятность, что строки совпадают.

### Синтаксис

```
ngramSimHash(string[, ngramsize])
```

### Аргументы

- `string` — строка. [String](#).
- `ngramsize` — размер n-грамм. Необязательный. Возможные значения: любое число от `1` до `25`. Значение по умолчанию: `3`. [UInt8](#).

### Возвращаемое значение

- Значение хеш-функции от строки.

Тип: [UInt64](#).

## Пример

Запрос:

```
SELECT ngramSimHash('ClickHouse') AS Hash;
```

Результат:

Hash
1627567969

## ngramSimHashCaseInsensitive

Выделяет из ASCII строки отрезки (n-граммы) размером `ngramsize` символов и возвращает n-граммовый `simhash`. Функция регистронезависимая.

Может быть использована для проверки двух строк на схожесть вместе с функцией [bitHammingDistance](#). Чем меньше [расстояние Хэмминга](#) между результатом вычисления `simhash` двух строк, тем больше вероятность, что строки совпадают.

## Синтаксис

```
ngramSimHashCaseInsensitive(string[, ngramsize])
```

## Аргументы

- `string` — строка. [String](#).
- `ngramsize` — размер n-грамм. Необязательный. Возможные значения: любое число от 1 до 25. Значение по умолчанию: 3. [UInt8](#).

## Возвращаемое значение

- Значение хеш-функции от строки.

Тип: [UInt64](#).

## Пример

Запрос:

```
SELECT ngramSimHashCaseInsensitive('ClickHouse') AS Hash;
```

Результат:

Hash
562180645

## ngramSimHashUTF8

Выделяет из UTF-8 строки отрезки (n-граммы) размером `ngramsize` символов и возвращает n-граммовый `simhash`. Функция регистрозависимая.

Может быть использована для проверки двух строк на схожесть вместе с функцией `bitHammingDistance`. Чем меньше **расстояние Хэмминга** между результатом вычисления `simhash` двух строк, тем больше вероятность, что строки совпадают.

## Синтаксис

```
ngramSimHashUTF8(string[, ngramsize])
```

## Аргументы

- `string` — строка. [String](#).
- `ngramsize` — размер n-грамм. Необязательный. Возможные значения: любое число от `1` до `25`. Значение по умолчанию: `3`. [UInt8](#).

## Возвращаемое значение

- Значение хеш-функции от строки.

Тип: [UInt64](#).

## Пример

Запрос:

```
SELECT ngramSimHashUTF8('ClickHouse') AS Hash;
```

Результат:

Hash
1628157797

## ngramSimHashCaseInsensitiveUTF8

Выделяет из UTF-8 строки отрезки (n-граммы) размером `ngramsize` символов и возвращает n-граммовый `simhash`. Функция регистронезависимая.

Может быть использована для проверки двух строк на схожесть вместе с функцией `bitHammingDistance`. Чем меньше **расстояние Хэмминга** между результатом вычисления `simhash` двух строк, тем больше вероятность, что строки совпадают.

## Синтаксис

```
ngramSimHashCaseInsensitiveUTF8(string[, ngramsize])
```

## Аргументы

- `string` — строка. [String](#).
- `ngramsize` — размер n-грамм. Необязательный. Возможные значения: любое число от `1` до `25`. Значение по умолчанию: `3`. [UInt8](#).

## Возвращаемое значение

- Значение хеш-функции от строки.

Тип: [UInt64](#).

## Пример

Запрос:

```
SELECT ngramSimHashCaseInsensitiveUTF8('ClickHouse') AS Hash;
```

Результат:

Hash
1636742693

## wordShingleSimHash

Выделяет из ASCII строки отрезки (шинглы) из `shinglesize` слов и возвращает шингловый `simhash`. Функция регистрозависимая.

Может быть использована для проверки двух строк на схожесть вместе с функцией `bitHammingDistance`. Чем меньше [расстояние Хэмминга](#) между результатом вычисления `simhash` двух строк, тем больше вероятность, что строки совпадают.

## Синтаксис

```
wordShingleSimHash(string[, shinglesize])
```

## Аргументы

- `string` — строка. [String](#).
- `shinglesize` — размер словесных шинглов. Необязательный. Возможные значения: любое число от 1 до 25. Значение по умолчанию: 3. [UInt8](#).

## Возвращаемое значение

- Значение хеш-функции от строки.

Тип: [UInt64](#).

## Пример

Запрос:

```
SELECT wordShingleSimHash('ClickHouse® is a column-oriented database management system (DBMS) for online analytical processing of queries (OLAP).') AS Hash;
```

Результат:

Hash
2328277067

## wordShingleSimHashCaseInsensitive

Выделяет из ASCII строки отрезки (шинглы) из `shinglesize` слов и возвращает шингловый `simhash`.  
Функция регистронезависимая.

Может быть использована для проверки двух строк на схожесть вместе с функцией `bitHammingDistance`. Чем меньше **расстояние Хэмминга** между результатом вычисления `simhash` двух строк, тем больше вероятность, что строки совпадают.

## Синтаксис

```
wordShingleSimHashCaseInsensitive(string[, shinglesize])
```

## Аргументы

- `string` — строка. [String](#).
- `shinglesize` — размер словесных шинглов. Необязательный. Возможные значения: любое число от 1 до 25. Значение по умолчанию: 3. [UInt8](#).

## Возвращаемое значение

- Значение хеш-функции от строки.

Тип: [UInt64](#).

## Пример

Запрос:

```
SELECT wordShingleSimHashCaseInsensitive('ClickHouse® is a column-oriented database management system  
(DBMS) for online analytical processing of queries (OLAP).') AS Hash;
```

Результат:

```
Hash  
2194812424
```

## wordShingleSimHashUTF8

Выделяет из UTF-8 строки отрезки (шинглы) из `shinglesize` слов и возвращает шингловый `simhash`.  
Функция регистрозависимая.

Может быть использована для проверки двух строк на схожесть вместе с функцией `bitHammingDistance`. Чем меньше **расстояние Хэмминга** между результатом вычисления `simhash` двух строк, тем больше вероятность, что строки совпадают.

## Синтаксис

```
wordShingleSimHashUTF8(string[, shinglesize])
```

## Аргументы

- `string` — строка. [String](#).
- `shinglesize` — размер словесных шинглов. Необязательный. Возможные значения: любое число от 1 до 25. Значение по умолчанию: 3. [UInt8](#).

## Возвращаемое значение

- Значение хеш-функции от строки.

Тип: [UInt64](#).

## Пример

Запрос:

```
SELECT wordShingleSimHashUTF8('ClickHouse® is a column-oriented database management system (DBMS) for online analytical processing of queries (OLAP).') AS Hash;
```

Результат:

Hash
2328277067

## wordShingleSimHashCaseInsensitiveUTF8

Выделяет из UTF-8 строки отрезки (шинглы) из shinglesize слов и возвращает шингловый simhash. Функция регистронезависимая.

Может быть использована для проверки двух строк на схожесть вместе с функцией [bitHammingDistance](#). Чем меньше [расстояние Хэмминга](#) между результатом вычисления simhash двух строк, тем больше вероятность, что строки совпадают.

## Синтаксис

```
wordShingleSimHashCaseInsensitiveUTF8(string[, shinglesize])
```

## Аргументы

- `string` — строка. [String](#).
- `shinglesize` — размер словесных шинглов. Необязательный. Возможные значения: любое число от 1 до 25. Значение по умолчанию: 3. [UInt8](#).

## Возвращаемое значение

- Значение хеш-функции от строки.

Тип: [UInt64](#).

## Пример

Запрос:

```
SELECT wordShingleSimHashCaseInsensitiveUTF8('ClickHouse® is a column-oriented database management system (DBMS) for online analytical processing of queries (OLAP).') AS Hash;
```

Результат:

Hash
2194812424

## ngramMinHash

Выделяет из ASCII строки отрезки (n-граммы) размером `ngramsize` символов и вычисляет хеш для каждой n-граммы. Использует `hashnum` минимальных хешей, чтобы вычислить минимальный хеш, и `hashnum` максимальных хешей, чтобы вычислить максимальный хеш. Возвращает кортеж из этих хешей. Функция регистровависимая.

Может быть использована для проверки двух строк на схожесть вместе с функцией `tupleHammingDistance`. Если для двух строк минимальные или максимальные хеши одинаковы, мы считаем, что эти строки совпадают.

## Синтаксис

```
ngramMinHash(string[, ngramsize, hashnum])
```

## Аргументы

- `string` — строка. [String](#).
- `ngramsize` — размер n-грамм. Необязательный. Возможные значения: любое число от 1 до 25. Значение по умолчанию: 3. [UInt8](#).
- `hashnum` — количество минимальных и максимальных хешей, которое используется при вычислении результата. Необязательный. Возможные значения: любое число от 1 до 25. Значение по умолчанию: 6. [UInt8](#).

## Возвращаемое значение

- Кортеж с двумя хешами — минимальным и максимальным.

Тип: [Tuple\(UInt64, UInt64\)](#).

## Пример

Запрос:

```
SELECT ngramMinHash('ClickHouse') AS Tuple;
```

Результат:

```
Tuple
(18333312859352735453,9054248444481805918)
```

## ngramMinHashCaseInsensitive

Выделяет из ASCII строки отрезки (n-граммы) размером `ngramsize` символов и вычисляет хеш для каждой n-граммы. Использует `hashnum` минимальных хешей, чтобы вычислить минимальный хеш, и `hashnum` максимальных хешей, чтобы вычислить максимальный хеш. Возвращает кортеж из этих хешей. Функция регистрационезависимая.

Может быть использована для проверки двух строк на схожесть вместе с функцией `tupleHammingDistance`. Если для двух строк минимальные или максимальные хеши одинаковы, мы считаем, что эти строки совпадают.

## Синтаксис

```
ngramMinHashCaseInsensitive(string[, ngramsize, hashnum])
```

## Аргументы

- `string` — строка. [String](#).
- `ngramsize` — размер n-грамм. Необязательный. Возможные значения: любое число от 1 до 25. Значение по умолчанию: 3. [UInt8](#).
- `hashnum` — количество минимальных и максимальных хешей, которое используется при вычислении результата. Необязательный. Возможные значения: любое число от 1 до 25. Значение по умолчанию: 6. [UInt8](#).

## Возвращаемое значение

- Кортеж с двумя хешами — минимальным и максимальным.

Тип: [Tuple\(UInt64, UInt64\)](#).

## Пример

Запрос:

```
SELECT ngramMinHashCaseInsensitive('ClickHouse') AS Tuple;
```

Результат:

```
Tuple
(2106263556442004574,13203602793651726206)
```

## ngramMinHashUTF8

Выделяет из UTF-8 строки отрезки (n-граммы) размером `ngramsize` символов и вычисляет хеш для каждой n-граммы. Использует `hashnum` минимальных хешей, чтобы вычислить минимальный хеш, и `hashnum` максимальных хешей, чтобы вычислить максимальный хеш. Возвращает кортеж из этих хешей. Функция регистрозависимая.

Может быть использована для проверки двух строк на схожесть вместе с функцией [tupleHammingDistance](#). Если для двух строк минимальные или максимальные хеши одинаковы, мы считаем, что эти строки совпадают.

## Синтаксис

```
ngramMinHashUTF8(string[, ngramsize, hashnum])
```

## Аргументы

- `string` — строка. [String](#).
- `ngramsize` — размер n-грамм. Необязательный. Возможные значения: любое число от 1 до 25. Значение по умолчанию: 3. [UInt8](#).
- `hashnum` — количество минимальных и максимальных хешей, которое используется при вычислении результата. Необязательный. Возможные значения: любое число от 1 до 25. Значение по умолчанию: 6. [UInt8](#).

## Возвращаемое значение

- Кортеж с двумя хешами — минимальным и максимальным.

Тип: Tuple(UInt64, UInt64).

## Пример

Запрос:

```
SELECT ngramMinHashUTF8('ClickHouse') AS Tuple;
```

Результат:

```
Tuple  
(18333312859352735453,6742163577938632877)
```

## ngramMinHashCaseInsensitiveUTF8

Выделяет из UTF-8 строки отрезки (п-граммы) размером `ngramsize` символов и вычисляет хеш для каждой п-граммы. Использует `hashnum` минимальных хешей, чтобы вычислить минимальный хеш, и `hashnum` максимальных хешей, чтобы вычислить максимальный хеш. Возвращает кортеж из этих хешей. Функция регистронезависимая.

Может быть использована для проверки двух строк на схожесть вместе с функцией `tupleHammingDistance`. Если для двух строк минимальные или максимальные хеши одинаковы, мы считаем, что эти строки совпадают.

## Синтаксис

```
ngramMinHashCaseInsensitiveUTF8(string [, ngramsize, hashnum])
```

## Аргументы

- `string` — строка. [String](#).
- `ngramsize` — размер п-грамм. Необязательный. Возможные значения: любое число от `1` до `25`. Значение по умолчанию: `3`. [UInt8](#).
- `hashnum` — количество минимальных и максимальных хешей, которое используется при вычислении результата. Необязательный. Возможные значения: любое число от `1` до `25`. Значение по умолчанию: `6`. [UInt8](#).

## Возвращаемое значение

- Кортеж с двумя хешами — минимальным и максимальным.

Тип: Tuple(UInt64, UInt64).

## Пример

Запрос:

```
SELECT ngramMinHashCaseInsensitiveUTF8('ClickHouse') AS Tuple;
```

Результат:

```
Tuple  
(12493625717655877135,13203602793651726206)
```

# ngramMinHashArg

Выделяет из ASCII строки отрезки (n-граммы) размером `ngramsize` символов и возвращает n-граммы с минимальным и максимальным хешами, вычисленными функцией `ngramMinHash` с теми же входными данными. Функция регистрозависимая.

## Синтаксис

```
ngramMinHashArg(string[, ngramsize, hashnum])
```

## Аргументы

- `string` — строка. [String](#).
- `ngramsize` — размер n-грамм. Необязательный. Возможные значения: любое число от `1` до `25`. Значение по умолчанию: `3`. [UInt8](#).
- `hashnum` — количество минимальных и максимальных хешей, которое используется при вычислении результата. Необязательный. Возможные значения: любое число от `1` до `25`. Значение по умолчанию: `6`. [UInt8](#).

## Возвращаемое значение

- Кортеж из двух кортежей, каждый из которых состоит из `hashnum` n-грамм.

Тип: [Tuple\(Tuple\(String\), Tuple\(String\)\)](#).

## Пример

Запрос:

```
SELECT ngramMinHashArg('ClickHouse') AS Tuple;
```

Результат:

```
Tuple--  
  ('ous','ick','lic','Hou','kHo','use'), ('Hou','lic','ick','ous','ckH','Cli')) |
```

# ngramMinHashArgCaseInsensitive

Выделяет из ASCII строки отрезки (n-граммы) размером `ngramsize` символов и возвращает n-граммы с минимальным и максимальным хешами, вычисленными функцией `ngramMinHashCaseInsensitive` с теми же входными данными. Функция регистронезависимая.

## Синтаксис

```
ngramMinHashArgCaseInsensitive(string[, ngramsize, hashnum])
```

## Аргументы

- `string` — строка. [String](#).
- `ngramsize` — размер n-грамм. Необязательный. Возможные значения: любое число от `1` до `25`. Значение по умолчанию: `3`. [UInt8](#).

- `hashnum` — количество минимальных и максимальных хешей, которое используется при вычислении результата. Необязательный. Возможные значения: любое число от 1 до 25. Значение по умолчанию: 6. `UInt8`.

## Возвращаемое значение

- Кортеж из двух кортежей, каждый из которых состоит из `hashnum` n-грамм.

Тип: `Tuple(Tuple(String), Tuple(String))`.

## Пример

Запрос:

```
SELECT ngramMinHashArgCaseInsensitive('ClickHouse') AS Tuple;
```

Результат:

```
└── Tuple
    └── ('ous','ick','lic','kHo','use','Cli'),('kHo','lic','ick','ous','ckH','Hou')
```

## ngramMinHashArgUTF8

Выделяет из UTF-8 строки отрезки (n-граммы) размером `ngramsize` символов и возвращает n-граммы с минимальным и максимальным хешами, вычисленными функцией `ngramMinHashUTF8` с теми же входными данными. Функция регистрозависимая.

## Синтаксис

```
ngramMinHashArgUTF8(string[, ngramsize, hashnum])
```

## Аргументы

- `string` — строка. `String`.
- `ngramsize` — размер n-грамм. Необязательный. Возможные значения: любое число от 1 до 25. Значение по умолчанию: 3. `UInt8`.
- `hashnum` — количество минимальных и максимальных хешей, которое используется при вычислении результата. Необязательный. Возможные значения: любое число от 1 до 25. Значение по умолчанию: 6. `UInt8`.

## Возвращаемое значение

- Кортеж из двух кортежей, каждый из которых состоит из `hashnum` n-грамм.

Тип: `Tuple(Tuple(String), Tuple(String))`.

## Пример

Запрос:

```
SELECT ngramMinHashArgUTF8('ClickHouse') AS Tuple;
```

Результат:

```
Tuple  
(({'ous','ick','lic','Hou','kHo','use'), ('kHo','Hou','lic','ick','ous','ckH')) |
```

## ngramMinHashArgCaseInsensitiveUTF8

Выделяет из UTF-8 строки отрезки (n-граммы) размером `ngramsize` символов и возвращает n-граммы с минимальным и максимальным хешами, вычисленными функцией `ngramMinHashCaseInsensitiveUTF8` с теми же входными данными. Функция регистронезависимая.

### Синтаксис

```
ngramMinHashArgCaseInsensitiveUTF8(string[, ngramsize, hashnum])
```

### Аргументы

- `string` — строка. [String](#).
- `ngramsize` — размер n-грамм. Необязательный. Возможные значения: любое число от `1` до `25`. Значение по умолчанию: `3`. [UInt8](#).
- `hashnum` — количество минимальных и максимальных хешей, которое используется при вычислении результата. Необязательный. Возможные значения: любое число от `1` до `25`. Значение по умолчанию: `6`. [UInt8](#).

### Возвращаемое значение

- Кортеж из двух кортежей, каждый из которых состоит из `hashnum` n-грамм.

Тип: [Tuple\(Tuple\(String\), Tuple\(String\)\)](#).

### Пример

Запрос:

```
SELECT ngramMinHashArgCaseInsensitiveUTF8('ClickHouse') AS Tuple;
```

Результат:

```
Tuple  
(({'ckH','ous','ick','lic','kHo','use'), ('kHo','lic','ick','ous','ckH','Hou')) |
```

## wordShingleMinHash

Выделяет из ASCII строки отрезки (шинглы) из `shinglesize` слов и вычисляет хеш для каждого шингла. Использует `hashnum` минимальных хешей, чтобы вычислить минимальный хеш, и `hashnum` максимальных хешей, чтобы вычислить максимальный хеш. Возвращает кортеж из этих хешей. Функция регистрозависимая.

Может быть использована для проверки двух строк на схожесть вместе с функцией `tupleHammingDistance`. Если для двух строк минимальные или максимальные хеши одинаковы, мы считаем, что эти строки совпадают.

### Синтаксис

```
wordShingleMinHash(string[, shinglesize, hashnum])
```

## Аргументы

- `string` — строка. [String](#).
- `shinglesize` — размер словесных шинглов. Необязательный. Возможные значения: любое число от `1` до `25`. Значение по умолчанию: `3`. [UInt8](#).
- `hashnum` — количество минимальных и максимальных хешей, которое используется при вычислении результата. Необязательный. Возможные значения: любое число от `1` до `25`. Значение по умолчанию: `6`. [UInt8](#).

## Возвращаемое значение

- Кортеж с двумя хешами — минимальным и максимальным.

Тип: [Tuple\(UInt64, UInt64\)](#).

## Пример

Запрос:

```
SELECT wordShingleMinHash('ClickHouse® is a column-oriented database management system (DBMS) for online analytical processing of queries (OLAP).') AS Tuple;
```

Результат:

```
Tuple
(16452112859864147620,5844417301642981317)
```

## wordShingleMinHashCaseInsensitive

Выделяет из ASCII строки отрезки (шинглы) из `shinglesize` слов и вычисляет хеш для каждого шингла. Использует `hashnum` минимальных хешей, чтобы вычислить минимальный хеш, и `hashnum` максимальных хешей, чтобы вычислить максимальный хеш. Возвращает кортеж из этих хешей. Функция регистронезависимая.

Может быть использована для проверки двух строк на схожесть вместе с функцией [tupleHammingDistance](#). Если для двух строк минимальные или максимальные хеши одинаковы, мы считаем, что эти строки совпадают.

## Синтаксис

```
wordShingleMinHashCaseInsensitive(string[, shinglesize, hashnum])
```

## Аргументы

- `string` — строка. [String](#).
- `shinglesize` — размер словесных шинглов. Необязательный. Возможные значения: любое число от `1` до `25`. Значение по умолчанию: `3`. [UInt8](#).
- `hashnum` — количество минимальных и максимальных хешей, которое используется при вычислении результата. Необязательный. Возможные значения: любое число от `1` до `25`. Значение по умолчанию: `6`. [UInt8](#).

## Возвращаемое значение

- Кортеж с двумя хешами — минимальным и максимальным.

Тип: `Tuple(UInt64, UInt64)`.

## Пример

Запрос:

```
SELECT wordShingleMinHashCaseInsensitive('ClickHouse® is a column-oriented database management system  
(DBMS) for online analytical processing of queries (OLAP).') AS Tuple;
```

Результат:

```
Tuple—  
[3065874883688416519,1634050779997673240]
```

## wordShingleMinHashUTF8

Выделяет из UTF-8 строки отрезки (шинглы) из `shinglesize` слов и вычисляет хеш для каждого шингла. Использует `hashnum` минимальных хешей, чтобы вычислить минимальный хеш, и `hashnum` максимальных хешей, чтобы вычислить максимальный хеш. Возвращает кортеж из этих хешей. Функция регистрозависимая.

Может быть использована для проверки двух строк на схожесть вместе с функцией `tupleHammingDistance`. Если для двух строк минимальные или максимальные хеши одинаковы, мы считаем, что эти строки совпадают.

## Синтаксис

```
wordShingleMinHashUTF8(string[, shinglesize, hashnum])
```

## Аргументы

- `string` — строка. [String](#).
- `shinglesize` — размер словесных шинглов. Необязательный. Возможные значения: любое число от 1 до 25. Значение по умолчанию: 3. [UInt8](#).
- `hashnum` — количество минимальных и максимальных хешей, которое используется при вычислении результата. Необязательный. Возможные значения: любое число от 1 до 25. Значение по умолчанию: 6. [UInt8](#).

## Возвращаемое значение

- Кортеж с двумя хешами — минимальным и максимальным.

Тип: `Tuple(UInt64, UInt64)`.

## Пример

Запрос:

```
SELECT wordShingleMinHashUTF8('ClickHouse® is a column-oriented database management system (DBMS) for  
online analytical processing of queries (OLAP).') AS Tuple;
```

Результат:

```
Tuple  
(16452112859864147620,5844417301642981317)
```

## wordShingleMinHashCaseInsensitiveUTF8

Выделяет из UTF-8 строки отрезки (шинглы) из `shinglesize` слов и вычисляет хеш для каждого шингла. Использует `hashnum` минимальных хешей, чтобы вычислить минимальный хеш, и `hashnum` максимальных хешей, чтобы вычислить максимальный хеш. Возвращает кортеж из этих хешей. Функция регистронезависимая.

Может быть использована для проверки двух строк на схожесть вместе с функцией `tupleHammingDistance`. Если для двух строк минимальные или максимальные хеши одинаковы, мы считаем, что эти строки совпадают.

### Синтаксис

```
wordShingleMinHashCaseInsensitiveUTF8(string[, shinglesize, hashnum])
```

### Аргументы

- `string` — строка. [String](#).
- `shinglesize` — размер словесных шинглов. Необязательный. Возможные значения: любое число от 1 до 25. Значение по умолчанию: 3. [UInt8](#).
- `hashnum` — количество минимальных и максимальных хешей, которое используется при вычислении результата. Необязательный. Возможные значения: любое число от 1 до 25. Значение по умолчанию: 6. [UInt8](#).

### Возвращаемое значение

- Кортеж с двумя хешами — минимальным и максимальным.

Тип: [Tuple\(UInt64, UInt64\)](#).

### Пример

Запрос:

```
SELECT wordShingleMinHashCaseInsensitiveUTF8('ClickHouse® is a column-oriented database management system  
(DBMS) for online analytical processing of queries (OLAP).') AS Tuple;
```

Результат:

```
Tuple  
(3065874883688416519,1634050779997673240)
```

## wordShingleMinHashArg

Выделяет из ASCII строки отрезки (шинглы) из `shinglesize` слов и возвращает шинглы с минимальным и максимальным хешами, вычисленными функцией [wordShingleMinHash](#) с теми же входными данными. Функция регистрозависимая.

## Синтаксис

```
wordShingleMinHashArg(string[, shinglesize, hashnum])
```

## Аргументы

- `string` — строка. [String](#).
- `shinglesize` — размер словесных шинглов. Необязательный. Возможные значения: любое число от 1 до 25. Значение по умолчанию: 3. [UInt8](#).
- `hashnum` — количество минимальных и максимальных хешей, которое используется при вычислении результата. Необязательный. Возможные значения: любое число от 1 до 25. Значение по умолчанию: 6. [UInt8](#).

## Возвращаемое значение

- Кортеж из двух кортежей, каждый из которых состоит из `hashnum` шинглов.

Тип: [Tuple\(Tuple\(String\), Tuple\(String\)\)](#).

## Пример

Запрос:

```
SELECT wordShingleMinHashArg('ClickHouse® is a column-oriented database management system (DBMS) for online analytical processing of queries (OLAP).', 1, 3) AS Tuple;
```

Результат:

```
Tuple-  
((‘OLAP’, ‘database’, ‘analytical’), (‘online’, ‘oriented’, ‘processing’)) |
```

## wordShingleMinHashArgCaseInsensitive

Выделяет из ASCII строки отрезки (шинглы) из `shinglesize` слов и возвращает шинглы с минимальным и максимальным хешами, вычисленными функцией [wordShingleMinHashCaseInsensitive](#) с теми же входными данными. Функция регистронезависимая.

## Синтаксис

```
wordShingleMinHashArgCaseInsensitive(string[, shinglesize, hashnum])
```

## Аргументы

- `string` — строка. [String](#).
- `shinglesize` — размер словесных шинглов. Необязательный. Возможные значения: любое число от 1 до 25. Значение по умолчанию: 3. [UInt8](#).
- `hashnum` — количество минимальных и максимальных хешей, которое используется при вычислении результата. Необязательный. Возможные значения: любое число от 1 до 25. Значение по умолчанию: 6. [UInt8](#).

## Возвращаемое значение

- Кортеж из двух кортежей, каждый из которых состоит из `hashnum` шинглов.

Тип: `Tuple(Tuple(String), Tuple(String))`.

## Пример

Запрос:

```
SELECT wordShingleMinHashArgCaseInsensitive('ClickHouse® is a column-oriented database management system  
(DBMS) for online analytical processing of queries (OLAP).', 1, 3) AS Tuple;
```

Результат:

```
Tuple—  
((queries','database','analytical'),('oriented','processing','DBMS')) |
```

## wordShingleMinHashArgUTF8

Выделяет из UTF-8 строки отрезки (шинглы) из `shinglesize` слов и возвращает шинглы с минимальным и максимальным хешами, вычисленными функцией `wordShingleMinHashUTF8` с теми же входными данными. Функция регистрозависимая.

## Синтаксис

```
wordShingleMinHashArgUTF8(string[, shinglesize, hashnum])
```

## Аргументы

- `string` — строка. [String](#).
- `shinglesize` — размер словесных шинглов. Необязательный. Возможные значения: любое число от 1 до 25. Значение по умолчанию: 3. [UInt8](#).
- `hashnum` — количество минимальных и максимальных хешей, которое используется при вычислении результата. Необязательный. Возможные значения: любое число от 1 до 25. Значение по умолчанию: 6. [UInt8](#).

## Возвращаемое значение

- Кортеж из двух кортежей, каждый из которых состоит из `hashnum` шинглов.

Тип: `Tuple(Tuple(String), Tuple(String))`.

## Пример

Запрос:

```
SELECT wordShingleMinHashArgUTF8('ClickHouse® is a column-oriented database management system (DBMS) for  
online analytical processing of queries (OLAP).', 1, 3) AS Tuple;
```

Результат:

```
Tuple—  
((OLAP','database','analytical'),('online','oriented','processing')) |
```

## wordShingleMinHashArgCaseInsensitiveUTF8

Выделяет из UTF-8 строки отрезки (шинглы) из `shinglesize` слов и возвращает шинглы с минимальным и максимальным хешами, вычисленными функцией `wordShingleMinHashCaseInsensitiveUTF8` с теми же входными данными. Функция регистронезависимая.

## Синтаксис

```
wordShingleMinHashArgCaseInsensitiveUTF8(string[, shinglesize, hashnum])
```

## Аргументы

- `string` — строка. `String`.
- `shinglesize` — размер словесных шинглов. Необязательный. Возможные значения: любое число от 1 до 25. Значение по умолчанию: 3. `UInt8`.
- `hashnum` — количество минимальных и максимальных хешей, которое используется при вычислении результата. Необязательный. Возможные значения: любое число от 1 до 25. Значение по умолчанию: 6. `UInt8`.

## Возвращаемое значение

- Кортеж из двух кортежей, каждый из которых состоит из `hashnum` шинглов.

Тип: `Tuple(Tuple(String), Tuple(String))`.

## Пример

Запрос:

```
SELECT wordShingleMinHashArgCaseInsensitiveUTF8('ClickHouse® is a column-oriented database management system (DBMS) for online analytical processing of queries (OLAP).', 1, 3) AS Tuple;
```

Результат:

```
Tuple-  
((queries', 'database', 'analytical'), ('oriented', 'processing', 'DBMS')) |
```

# ФУНКЦИИ ГЕНЕРАЦИИ ПСЕВДОСЛУЧАЙНЫХ ЧИСЕЛ

Используются криптографические генераторы псевдослучайных чисел.

Все функции принимают ноль аргументов или один аргумент.

В случае, если передан аргумент - он может быть любого типа, и его значение никак не используется.

Этот аргумент нужен только для того, чтобы предотвратить склейку одинаковых выражений - чтобы две разные записи одной функции возвращали разные столбцы, с разными случайными числами.

## rand

Возвращает псевдослучайное число типа `UInt32`, равномерно распределённое среди всех чисел типа `UInt32`.

Используется linear congruential generator.

## rand64

Возвращает псевдослучайное число типа UInt64, равномерно распределённое среди всех чисел типа UInt64.

Используется linear congruent generator.

## randConstant

Создает константный столбец с псевдослучайным значением.

### Синтаксис

```
randConstant([x])
```

### Аргументы

- **x** — выражение, возвращающее значение одного из поддерживаемых типов данных. Значение используется, чтобы избежать склейки одинаковых выражений, если функция вызывается несколько раз в одном запросе. Необязательный параметр.

### Возвращаемое значение

- Псевдослучайное число.

Тип: UInt32.

### Пример

Запрос:

```
SELECT rand(), rand(1), rand(number), randConstant(), randConstant(1), randConstant(number)  
FROM numbers(3)
```

Результат:

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

## Случайные функции для работы со строками

### randomString

### randomFixedString

### randomPrintableASCII

### randomStringUTF8

### fuzzBits

### Синтаксис

```
fuzzBits([s], [prob])
```

Инвертирует каждый бит `s` с вероятностью `prob`.

## Аргументы

- `s` — `String` or `FixedString`
- `prob` — constant `Float32/64`

## Возвращаемое значение

Измененная случайным образом строка с тем же типом, что и `s`.

## Пример

Запрос:

```
SELECT fuzzBits(materialize('abacaba'), 0.1)
FROM numbers(3)
```

Результат:

```
fuzzBits(materialize('abacaba'), 0.1) ↴
abaaaja
a*cjab+
aecaa2A
```

# ФУНКЦИИ КОДИРОВАНИЯ

## char

Возвращает строку, длина которой равна числу переданных аргументов, и каждый байт имеет значение соответствующего аргумента. Принимает несколько числовых аргументов. Если значение аргумента выходит за диапазон `UInt8` (0..255), то оно преобразуется в `UInt8` с возможным округлением и переполнением.

## Синтаксис

```
char(number_1, [number_2, ..., number_n]);
```

## Аргументы

- `number_1, number_2, ..., number_n` — числовые аргументы, которые интерпретируются как целые числа. Типы: `Int`, `Float`.

## Возвращаемое значение

- Стока из соответствующих байт.

Тип: `String`.

## Пример

Запрос:

```
SELECT char(104.1, 101, 108.9, 108.9, 111) AS hello;
```

Результат:

```
hello  
hello |
```

Вы можете создать строку в произвольной кодировке, передав соответствующие байты. Пример для UTF-8:

Запрос:

```
SELECT char(0xD0, 0xBF, 0xD1, 0x80, 0xD0, 0xB8, 0xD0, 0xB2, 0xD0, 0xB5, 0xD1, 0x82) AS hello;
```

Результат:

```
hello  
привет |
```

Запрос:

```
SELECT char(0xE4, 0xBD, 0xA0, 0xE5, 0xA5, 0xBD) AS hello;
```

Результат:

```
hello  
你好 |
```

## hex

Возвращает строку, содержащую шестнадцатеричное представление аргумента.

Синоним: HEX.

### Синтаксис

```
hex(arg)
```

Функция использует прописные буквы A-F и не использует никаких префиксов (например, 0x) или суффиксов (например, h).

Для целочисленных аргументов возвращает шестнадцатеричные цифры от наиболее до наименее значимых (big endian, человекочитаемый порядок). Он начинается с самого значимого ненулевого байта (начальные нулевые байты опущены), но всегда выводит обе цифры каждого байта, даже если начальная цифра равна нулю.

Значения типа [Date](#) и [DateTime](#) формируются как соответствующие целые числа (количество дней с момента Unix-эпохи для Date и значение Unix Timestamp для DateTime).

Для [String](#) и [FixedString](#), все байты просто кодируются как два шестнадцатеричных числа. Нулевые байты не опущены.

Значения **Float** и **Decimal** кодируются как их представление в памяти. Поскольку ClickHouse поддерживает архитектуру `little-endian`, они кодируются от младшего к старшему байту. Нулевые начальные/конечные байты не опущены.

## Аргументы

- `arg` — значение для преобразования в шестнадцатеричное. **String**, **UInt**, **Float**, **Decimal**, **Date** или **DateTime**.

## Возвращаемое значение

- Стока — шестнадцатеричное представление аргумента.

Тип: **String**.

## Примеры

Запрос:

```
SELECT hex(1);
```

Результат:

```
01
```

Запрос:

```
SELECT hex(toFloat32(number)) AS hex_presentation FROM numbers(15, 2);
```

Результат:

hex_presentation
00007041
00008041

Запрос:

```
SELECT hex(toFloat64(number)) AS hex_presentation FROM numbers(15, 2);
```

Результат:

hex_presentation
0000000000002E40
0000000000003040

## unhex(str)

Выполняет операцию, обратную **hex**. Функция интерпретирует каждую пару шестнадцатеричных цифр аргумента как число и преобразует его в символ. Возвращаемое значение представляет собой двоичную строку (BLOB).

Если вы хотите преобразовать результат в число, вы можете использовать функции **reverse** и **reinterpretAs**.

## Примечание

Если `unhex` вызывается из `clickhouse-client`, двоичные строки отображаются с использованием UTF-8.

Синоним: UNHEX.

### Синтаксис

```
unhex(arg)
```

### Аргументы

- `arg` — Стока, содержащая любое количество шестнадцатеричных цифр. Тип: `String`.

Поддерживаются как прописные, так и строчные буквы A-F. Количество шестнадцатеричных цифр не обязательно должно быть четным. Если оно нечетное, последняя цифра интерпретируется как наименее значимая половина байта 00-0F. Если строка аргумента содержит что-либо, кроме шестнадцатеричных цифр, возвращается некоторый результат, определенный реализацией (исключение не создается).

### Возвращаемое значение

- Бинарная строка (BLOB).

Тип: `String`.

### Пример

Запрос:

```
SELECT unhex('303132'), UNHEX('4D7953514C');
```

Результат:

unhex('303132')	unhex('4D7953514C')
012	MySQL

Запрос:

```
SELECT reinterpretAsUInt64(reverse(unhex('FFF'))) AS num;
```

Результат:

num
4095

## bin

Возвращает строку, содержащую бинарное представление аргумента.

### Синтаксис

```
bin(arg)
```

Синоним: `BIN`.

Для целочисленных аргументов возвращаются двоичные числа от наиболее значимого до наименее значимого (`big-endian`, человекочитаемый порядок). Порядок начинается с самого значимого ненулевого байта (начальные нулевые байты опущены), но всегда возвращает восемь цифр каждого байта, если начальная цифра равна нулю.

Значения типа `Date` и `DateTime` формируются как соответствующие целые числа (количество дней с момента Unix-эпохи для `Date` и значение Unix Timestamp для `DateTime`).

Для `String` и `FixedString` все байты кодируются как восемь двоичных чисел. Нулевые байты не опущены.

Значения `Float` и `Decimal` кодируются как их представление в памяти. Поскольку ClickHouse поддерживает архитектуру `little-endian`, они кодируются от младшего к старшему байту. Нулевые начальные/конечные байты не опущены.

## Аргументы

- `arg` — значение для преобразования в двоичный код. `String`, `FixedString`, `UInt`, `Float`, `Decimal`, `Date` или `DateTime`.

## Возвращаемое значение

- Бинарная строка (BLOB) — двоичное представление аргумента.

Тип: `String`.

## Примеры

Запрос:

```
SELECT bin(14);
```

Результат:

```
bin(14)
00001110 |
```

Запрос:

```
SELECT bin(toFloat32(number)) AS bin_presentation FROM numbers(15, 2);
```

Результат:

```
bin_presentation
0000000000000000111000001000001
0000000000000000100000001000001 |
```

Запрос:

```
SELECT bin(toFloat64(number)) AS bin_presentation FROM numbers(15, 2);
```

## Результат:

unbin

Интерпретирует каждую пару двоичных цифр аргумента как число и преобразует его в байт, представленный числом. Функция выполняет операцию, противоположную `bin`.

## **Синтаксис**

unbin(arg)

Синоним: UNBIN.

Для числового аргумента `unbin()` не возвращает значение, обратное результату `bin()`. Чтобы преобразовать результат в число, используйте функции `reverse` и `reinterpretAs`.

## Примечание

Если `unbin` вызывается из клиента `clickhouse-client`, бинарная строка возвращается в кодировке UTF-8.

Поддерживает двоичные цифры 0 и 1. Количество двоичных цифр не обязательно должно быть кратно восьми. Если строка аргумента содержит что-либо, кроме двоичных цифр, возвращается некоторый результат, определенный реализацией (ошибки не возникает).

## Аргументы

- `arg` — строка, содержащая любое количество двоичных цифр. `String`.

## **Возвращаемое значение**

- Бинарная строка (BLOB).

Тип: String.

## Примеры

## Запрос:

```
SELECT UNBIN('001100000011000100110010'), UNBIN('010011010111001010100110101000101001100');
```

## Результат:

```
--unbin('001100000011000100110010')--unbin('0100110101111001010100110101000101001100')--  
012 | MySQL |
```

### Запрос:

```
SELECT reinterpretAsUInt64(reverse(unbin('1110'))) AS num;
```

Результат:

```
num
14 |
```

## UUIDStringToNum(str)

Принимает строку, содержащую 36 символов в формате `123e4567-e89b-12d3-a456-426655440000`, и возвращает в виде набора байт в `FixedString(16)`.

## UUIDNumToString(str)

Принимает значение типа `FixedString(16)`. Возвращает строку из 36 символов в текстовом виде.

## bitmaskToList(num)

Принимает целое число. Возвращает строку, содержащую список степеней двойки, в сумме дающих исходное число; по возрастанию, в текстовом виде, через запятую, без пробелов.

## bitmaskToArray(num)

Принимает целое число. Возвращает массив чисел типа `UInt64`, содержащий степени двойки, в сумме дающих исходное число; числа в массиве идут по возрастанию.

## bitPositionsToArray(num)

Принимает целое число и приводит его к беззнаковому виду. Возвращает массив `UInt64` чисел, который содержит список позиций битов `arg`, равных 1, в порядке возрастания.

### Синтаксис

```
bitPositionsToArray(arg)
```

### Аргументы

- `arg` — целое значение. [Int/UInt](#).

### Возвращаемое значение

- Массив, содержащий список позиций битов, равных 1, в порядке возрастания.

Тип: [Array\(UInt64\)](#).

### Примеры

Запрос:

```
SELECT bitPositionsToArray(toInt8(1)) AS bit_positions;
```

Результат:

```
bit_positions
[0] |
```

Запрос:

```
SELECT bitPositionsToArray(tolnt8(-1)) AS bit_positions;
```

Результат:

```
bit_positions  
[0,1,2,3,4,5,6,7] |
```

## Функции для работы с UUID

### generateUUIDv4

Генерирует идентификатор [UUID версии 4](#).

```
generateUUIDv4()
```

#### Возвращаемое значение

Значение типа [UUID](#).

#### Пример использования

Этот пример демонстрирует, как создать таблицу с UUID-колонкой и добавить в нее сгенерированный UUID.

```
CREATE TABLE t_uuid (x UUID) ENGINE=TinyLog  
INSERT INTO t_uuid SELECT generateUUIDv4()  
SELECT * FROM t_uuid
```

```
f4bf890f-f9dc-4332-ad5c-0c18e73f28e9 |
```

## empty

Проверяет, является ли входной UUID пустым.

#### Синтаксис

```
empty(UUID)
```

UUID считается пустым, если он содержит все нули (нулевой UUID).

Функция также поддерживает работу с типами [Array](#) и [String](#).

#### Параметры

- `x` — UUID на входе функции. [UUID](#).

#### Возвращаемое значение

- Возвращает 1 для пустого UUID или 0 — для непустого UUID.

Тип: [UInt8](#).

## Пример

Для генерации UUID-значений предназначена функция [generateUUIDv4](#).

Запрос:

```
SELECT empty(generateUUIDv4());
```

Ответ:

```
empty(generateUUIDv4())  
0 |
```

## notEmpty

Проверяет, является ли входной UUID непустым.

### Синтаксис

```
notEmpty(UUID)
```

UUID считается пустым, если он содержит все нули (нулевой UUID).

Функция также поддерживает работу с типами [Array](#) и [String](#).

### Параметры

- `x` — UUID на входе функции. [UUID](#).

### Возвращаемое значение

- Возвращает `1` для непустого UUID или `0` — для пустого UUID.

Тип: [UInt8](#).

## Пример

Для генерации UUID-значений предназначена функция [generateUUIDv4](#).

Запрос:

```
SELECT notEmpty(generateUUIDv4());
```

Результат:

```
notEmpty(generateUUIDv4())  
1 |
```

## toUUID (x)

Преобразует значение типа [String](#) в тип UUID.

```
toUUID(String)
```

## **Возвращаемое значение**

Значение типа UUID.

## **Пример использования**

```
SELECT toUUID('61f0c404-5cb3-11e7-907b-a6006ad3dba0') AS uuid
```

61f0c404-5cb3-11e7-907b-a6006ad3dba0 |  
  ↑  
  uuid

## **toUUIDOrNull (x)**

Принимает строку, и пытается преобразовать в тип UUID. При неудаче возвращает NULL.

```
toUUIDOrNull(String)
```

## **Возвращаемое значение**

Значение типа Nullable(UUID).

## **Пример использования**

```
SELECT toUUIDOrNull('61f0c404-5cb3-11e7-907b-a6006ad3dba0T') AS uuid
```

  ↑  
  uuid  
  NULL |

## **toUUIDOrZero (x)**

Принимает строку, и пытается преобразовать в тип UUID. При неудаче возвращает нулевой UUID.

```
toUUIDOrZero(String)
```

## **Возвращаемое значение**

Значение типа UUID.

## **Пример использования**

```
SELECT toUUIDOrZero('61f0c404-5cb3-11e7-907b-a6006ad3dba0T') AS uuid
```

  ↑  
  00000000-0000-0000-000000000000 |  
  uuid

## **UUIDStringToNum**

Принимает строку, содержащую 36 символов в формате xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx, и возвращает в виде набора байт в [FixedSize\(16\)](#).

```
UUIDStringToNum(String)
```

## Возвращаемое значение

FixedString(16)

## Пример использования

```
SELECT  
    '612f3c40-5d3b-217e-707b-6a546a3d7b29' AS uuid,  
    UUIDStringToNum(uuid) AS bytes
```

```
uuid——— bytes———  
612f3c40-5d3b-217e-707b-6a546a3d7b29 | a/<@];!~p{jTj={}) |
```

## UUIDNumToString

Принимает значение типа [FixedString\(16\)](#). Возвращает строку из 36 символов в текстовом виде.

```
UUIDNumToString(FixedString(16))
```

## Возвращаемое значение

Значение типа String.

## Пример использования

```
SELECT  
    'a/<@];!~p{jTj={})' AS bytes,  
    UUIDNumToString(toFixedString(bytes, 16)) AS uuid
```

```
bytes——— uuid———  
a/<@];!~p{jTj={}) | 612f3c40-5d3b-217e-707b-6a546a3d7b29 |
```

## serverUUID()

Возвращает случайный и уникальный UUID, который генерируется при первом запуске сервера и сохраняется навсегда. Результат записывается в файл `uuid`, расположенный в каталоге сервера ClickHouse `/var/lib/clickhouse/`.

## Синтаксис

```
serverUUID()
```

## Возвращаемое значение

- UUID сервера.

Тип: [UUID](#).

## См. также:

- [dictGetUUID](#)

- [dictGetUUIDOrDefault](#)

## ФУНКЦИИ ДЛЯ РАБОТЫ С URL

Все функции работают не по RFC - то есть, максимально упрощены ради производительности.

### ФУНКЦИИ, ИЗВЛЕКАЮЩИЕ ЧАСТЬ URL-А

Если в URL-е нет ничего похожего, то возвращается пустая строка.

#### protocol

Возвращает протокол. Примеры: http, ftp, mailto, magnet...

#### domain

Извлекает имя хоста из URL.

```
domain(url)
```

#### Аргументы

- url — URL. Тип — [String](#).

URL может быть указан со схемой или без неё. Примеры:

```
svn+ssh://some.svn-hosting.com:80/repo/trunk  
some.svn-hosting.com:80/repo/trunk  
https://yandex.com/time/
```

Для указанных примеров функция `domain` возвращает следующие результаты:

```
some.svn-hosting.com  
some.svn-hosting.com  
yandex.com
```

#### Возвращаемые значения

- Имя хоста. Если ClickHouse может распарсить входную строку как URL.
- Пустая строка. Если ClickHouse не может распарсить входную строку как URL.

Тип — [String](#).

#### Пример

```
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

Возвращает домен, удалив префикс 'www.', если он присутствовал.

## topLevelDomain

Извлекает домен верхнего уровня из URL.

```
topLevelDomain(url)
```

### Аргументы

- `url` — URL. Тип — [String](#).

URL может быть указан со схемой или без неё. Примеры:

```
svn+ssh://some.svn-hosting.com:80/repo/trunk  
some.svn-hosting.com:80/repo/trunk  
https://yandex.com/time/
```

### Возвращаемые значения

- Имя домена. Если ClickHouse может распарсить входную строку как URL.
- Пустая строка. Если ClickHouse не может распарсить входную строку как URL.

Тип — [String](#).

### Пример

```
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

Возвращает «первый существенный поддомен». Это понятие является нестандартным и специфично для Яндекс.Метрики. Первый существенный поддомен — это домен второго уровня, если он не равен одному из com, net, org, со, или домен третьего уровня, иначе. Например, `firstSignificantSubdomain('https://news.yandex.ru/')` = 'yandex', `firstSignificantSubdomain('https://news.yandex.com.tr/')` = 'yandex'. Список «несущественных» доменов второго уровня и другие детали реализации могут изменяться в будущем.

## cutToFirstSignificantSubdomain

Возвращает часть домена, включающую поддомены верхнего уровня до «первого существенного поддомена» (см. выше).

Например, `cutToFirstSignificantSubdomain('https://news.yandex.com.tr/')` = 'yandex.com.tr'.

## cutToFirstSignificantSubdomainCustom

Возвращает часть домена, включающую поддомены верхнего уровня до первого существенного поддомена. Принимает имя пользовательского [списка доменов верхнего уровня](#).

Полезно, если требуется актуальный список доменов верхнего уровня или если есть пользовательский.

Пример конфигурации:

```
<!-- <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>
```

## Синтаксис

```
cutToFirstSignificantSubdomain(URL, TLD)
```

## Аргументы

- URL — URL. [String](#).
- TLD — имя пользовательского списка доменов верхнего уровня. [String](#).

## Возвращаемое значение

- Часть домена, включающая поддомены верхнего уровня до первого существенного поддомена.

Тип: [String](#).

## Пример

Запрос:

```
SELECT cutToFirstSignificantSubdomainCustom('bar.foo.there-is-no-such-domain', 'public_suffix_list');
```

Результат:

```
cutToFirstSignificantSubdomainCustom('bar.foo.there-is-no-such-domain', 'public_suffix_list')—
foo.there-is-no-such-domain |
```

## Смотрите также

- [firstSignificantSubdomain](#).

## cutToFirstSignificantSubdomainCustomWithWWW

Возвращает часть домена, включающую поддомены верхнего уровня до первого существенного поддомена, не опуская "www". Принимает имя пользовательского списка доменов верхнего уровня.

Полезно, если требуется актуальный список доменов верхнего уровня или если есть пользовательский.

Пример конфигурации:

```
<!-- <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>
```

## Синтаксис

```
cutToFirstSignificantSubdomainCustomWithWWW(URL, TLD)
```

## Аргументы

- URL — URL. [String](#).
- TLD — имя пользовательского списка доменов верхнего уровня. [String](#).

## Возвращаемое значение

- Часть домена, включающая поддомены верхнего уровня до первого существенного поддомена, без удаления `www`.

Тип: [String](#).

## Пример

Запрос:

```
SELECT cutToFirstSignificantSubdomainCustomWithWWW('www.foo', 'public_suffix_list');
```

Результат:

```
└─cutToFirstSignificantSubdomainCustomWithWWW('www.foo', 'public_suffix_list')─┐  
   www.foo                                |
```

## Смотрите также

- [firstSignificantSubdomain](#).

## firstSignificantSubdomainCustom

Возвращает первый существенный поддомен. Принимает имя пользовательского списка доменов верхнего уровня.

Полезно, если требуется актуальный список доменов верхнего уровня или если есть пользовательский.

Пример конфигурации:

```
<!-- <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>
```

## Синтаксис

```
firstSignificantSubdomainCustom(URL, TLD)
```

## Аргументы

- URL — URL. [String](#).
- TLD — имя пользовательского списка доменов верхнего уровня. [String](#).

## **Возвращаемое значение**

- Первый существенный поддомен.

Тип: [String](#).

## **Пример**

Запрос:

```
SELECT firstSignificantSubdomainCustom('bar.foo.there-is-no-such-domain', 'public_suffix_list');
```

Результат:

```
firstSignificantSubdomainCustom('bar.foo.there-is-no-such-domain', 'public_suffix_list')—  
|  
foo
```

## **Смотрите также**

- [firstSignificantSubdomain](#).

## **port(URL[, default\_port = 0])**

Возвращает порт или значение `default_port`, если в URL-адресе нет порта (или передан невалидный URL)

## **path**

Возвращает путь. Пример: `/top/news.html` Путь не включает в себя query string.

## **pathFull**

То же самое, но включая query string и fragment. Пример: `/top/news.html?page=2#comments`

## **queryString**

Возвращает query-string. Пример: `page=1&lr=213`. query-string не включает в себя начальный знак вопроса, а также # и всё, что после #.

## **fragment**

Возвращает fragment identifier. fragment не включает в себя начальный символ решётки.

## **queryStringAndFragment**

Возвращает query string и fragment identifier. Пример: `страница=1#29390`.

## **extractURLParameter(URL, name)**

Возвращает значение параметра `name` в URL, если такой есть; или пустую строку, иначе; если параметров с таким именем много - вернуть первый попавшийся. Функция работает при допущении, что имя параметра закодировано в URL в точности таким же образом, что и в переданном аргументе.

## **extractURLParameters(URL)**

Возвращает массив строк вида `name=value`, соответствующих параметрам URL. Значения никак не декодируются.

## extractURLParameterNames(URL)

Возвращает массив строк вида name, соответствующих именам параметров URL. Значения никак не декодируются.

## URLHierarchy(URL)

Возвращает массив, содержащий URL, обрезанный с конца по символам /, ? в пути и query-string. Подряд идущие символы-разделители считаются за один. Резка производится в позиции после всех подряд идущих символов-разделителей. Пример:

## URLPathHierarchy(URL)

То же самое, но без протокола и хоста в результате. Элемент / (корень) не включается. Пример: Функция используется для реализации древовидных отчётов по URL в Яндекс.Метрике.

```
URLPathHierarchy('https://example.com/browse/CONV-6788') =  
[  
  '/browse/',  
  '/browse/CONV-6788'  
]
```

## decodeURLComponent(URL)

Возвращает декодированный URL.

Пример:

```
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

Извлекает сетевую локальность (username:password@host:port) из URL.

### Синтаксис

```
netloc(URL)
```

### Аргументы

- url — URL. Тип — [String](#).

### Возвращаемое значение

- username:password@host:port.

Тип: [String](#).

### Пример

Запрос:

```
SELECT netloc('http://paul@www.example.com:80/');
```

Результат:

```
netloc('http://paul@www.example.com:80/')—  
paul@www.example.com:80 |
```

## ФУНКЦИИ, УДАЛЯЮЩИЕ ЧАСТЬ ИЗ URL-А

Если в URL-е нет ничего похожего, то URL остаётся без изменений.

### **cutWWW**

Удаляет не более одного ‘www.’ с начала домена URL-а, если есть.

### **cutQueryString**

Удаляет query string. Знак вопроса тоже удаляется.

### **cutFragment**

Удаляет fragment identifier. Символ решётки тоже удаляется.

### **cutQueryStringAndFragment**

Удаляет query string и fragment identifier. Знак вопроса и символ решётки тоже удаляются.

### **cutURLParameter(URL, name)**

Удаляет параметр URL с именем name, если такой есть. Функция работает при допущении, что имя параметра закодировано в URL в точности таким же образом, что и в переданном аргументе.

## ФУНКЦИИ ДЛЯ РАБОТЫ С IP-АДРЕСАМИ

### **IPv4NumToString(num)**

Принимает число типа UInt32. Интерпретирует его, как IPv4-адрес в big endian. Возвращает строку, содержащую соответствующий IPv4-адрес в формате A.B.C.D (числа в десятичной форме через точки).

Синоним: INET\_NTOA.

### **IPv4StringToNum(s)**

Функция, обратная к IPv4NumToString. Если IPv4 адрес в неправильном формате, то возвращает 0.

Синоним: INET\_ATON.

### **IPv4NumToStringClassC(num)**

Похоже на IPv4NumToString, но вместо последнего октета используется xxx.

Пример:

```

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

В связи с тем, что использование xxx весьма необычно, это может быть изменено в дальнейшем.  
Вам не следует полагаться на конкретный вид этого фрагмента.

## IPv6NumToString(x)

Принимает значение типа FixedString(16), содержащее IPv6-адрес в бинарном виде. Возвращает строку, содержащую этот адрес в текстовом виде.

IPv6-mapped IPv4 адреса выводятся в формате ::ffff:111.222.33.44.

Примеры: INET6\_NTOA.

Примеры:

```

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('0000000000000000FFFF')
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:eeee	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

Функция, обратная к [IPv6NumToString](#). Если IPv6 адрес передан в неправильном формате, то возвращает строку из нулевых байт.

Если IP адрес является корректным IPv4 адресом, функция возвращает его IPv6 эквивалент.

HEX может быть в любом регистре.

Синоним: INET6\_ATON.

### Синтаксис

```
IPv6StringToNum(string)
```

### Аргумент

- string — IP адрес. [String](#).

### Возвращаемое значение

- Адрес IPv6 в двоичном представлении.

Тип: [FixedString\(16\)](#).

### Пример

Запрос:

```
SELECT addr, cutIPv6(IPv6StringToNum(addr), 0, 0) FROM (SELECT ['notaddress', '127.0.0.1', '1111::ffff'] AS addr)
ARRAY JOIN addr;
```

Результат:

addr	cutIPv6(IPv6StringToNum(addr), 0, 0)
notaddress	::
127.0.0.1	::ffff:127.0.0.1
1111::ffff	1111::ffff

## Смотрите также

- [cutIPv6](#).

## IPv4ToIPv6(x)

Принимает число типа UInt32. Интерпретирует его, как IPv4-адрес в **big endian**. Возвращает значение FixedString(16), содержащее адрес IPv6 в двоичном формате. Примеры:

```
SELECT IPv6NumToString(IPv4ToIPv6(IPv4StringToNum('192.168.0.1'))) AS addr;
```

addr  
::ffff:192.168.0.1 |

## cutIPv6(x, bytesToCutForIPv6, bytesToCutForIPv4)

Принимает значение типа FixedString(16), содержащее IPv6-адрес в бинарном виде. Возвращает строку, содержащую адрес из указанного количества байтов, удаленных в текстовом формате. Например:

```
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:0 | ::ffff:192.168.0.0 |

## IPv4CIDRToRange(ipv4, Cidr),

Принимает на вход IPv4 и значение UInt8, содержащее **CIDR**. Возвращает кортеж с двумя IPv4, содержащими нижний и более высокий диапазон подсети.

```
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),

Принимает на вход IPv6 и значение UInt8, содержащее CIDR. Возвращает кортеж с двумя IPv6, содержащими нижний и более высокий диапазон подсети.

```
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')

## toIPv4(string)

Псевдоним функции `IPv4StringToNum()` которая принимает строку с адресом IPv4 и возвращает значение типа `IPv4`, которое равно значению, возвращаемому функцией `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

Приводит строку с адресом в формате IPv6 к типу `IPv6`. Возвращает пустое значение, если входящая строка не является корректным IP адресом.

Похоже на функцию `IPv6StringToNum`, которая представляет адрес IPv6 в двоичном виде.

Если входящая строка содержит корректный IPv4 адрес, функция возвращает его IPv6 эквивалент.

### Синтаксис

```
toIPv6(string)
```

### Аргумент

- `string` — IP адрес. `String`

### Возвращаемое значение

- IP адрес.

Тип: `IPv6`.

### Примеры

Запрос:

```
WITH '2001:438:ffff::407d:1bc1' AS IPv6_string
SELECT
  hex(IPv6StringToNum(IPv6_string)),
  hex(toIPv6(IPv6_string));
```

Результат:

```
hex(IPv6StringToNum(IPv6_string)) └── hex(toIPv6(IPv6_string)) ──────────  
20010438FFFF000000000000407D1BC1 | 20010438FFFF000000000000407D1BC1 |
```

Запрос:

```
SELECT toIPv6('127.0.0.1');
```

Результат:

```
toIPv6('127.0.0.1') └  
::ffff:127.0.0.1 |
```

## isIPv4String

Определяет, является ли строка адресом IPv4 или нет. Также вернет 0, если string — адрес IPv6.

### Синтаксис

```
isIPv4String(string)
```

### Аргументы

- string — IP адрес. [String](#).

### Возвращаемое значение

- 1 если string является адресом IPv4 , иначе — 0.

Тип: [UInt8](#).

### Примеры

Запрос:

```
SELECT addr, isIPv4String(addr) FROM ( SELECT ['0.0.0.0', '127.0.0.1', '::ffff:127.0.0.1'] AS addr ) ARRAY JOIN addr;
```

Результат:

```
addr ────────── isIPv4String(addr) ──────────  
0.0.0.0 | 1 |  
127.0.0.1 | 1 |  
::ffff:127.0.0.1 | 0 |
```

## isIPv6String

Определяет, является ли строка адресом IPv6 или нет. Также вернет 0, если string — адрес IPv4.

### Синтаксис

```
isIPv6String(string)
```

### Аргументы

- `string` — IP адрес. [String](#).

## Возвращаемое значение

- 1 если `string` является адресом IPv6 , иначе — 0.

Тип: [UInt8](#).

## Примеры

Запрос:

```
SELECT addr, isIPv6String(addr) FROM ( SELECT [':', '1111::ffff', '::ffff:127.0.0.1', '127.0.0.1'] AS addr ) ARRAY JOIN addr;
```

Результат:

addr	isIPv6String(addr)
::	1
1111::ffff	1
::ffff:127.0.0.1	1
127.0.0.1	0

## isIPAddressInRange

Проверяет, попадает ли IP адрес в интервал, заданный в нотации [CIDR](#).

## Синтаксис

```
isIPAddressInRange(address, prefix)
```

Функция принимает IPv4 или IPv6 адрес виде строки. Возвращает 0, если версия адреса и интервала не совпадают.

## Аргументы

- `address` — IPv4 или IPv6 адрес. [String](#).
- `prefix` — IPv4 или IPv6 подсеть, заданная в нотации CIDR. [String](#).

## Возвращаемое значение

- 1 или 0.

Тип: [UInt8](#).

## Примеры

Запрос:

```
SELECT isIPAddressInRange('127.0.0.1', '127.0.0.0/8');
```

Результат:

isIPAddressInRange('127.0.0.1', '127.0.0.0/8')
1

Запрос:

```
SELECT isIPAddressInRange('127.0.0.1', 'ffff::/16');
```

Результат:

```
+-----+  
| isIPAddressInRange('127.0.0.1', 'ffff::/16') |  
+-----+  
0 |
```

## ФУНКЦИИ ДЛЯ РАБОТЫ С JSON

В Яндекс.Метрике пользователями передаётся JSON в качестве параметров визитов. Для работы с таким JSON-ом, реализованы некоторые функции. (Хотя в большинстве случаев, JSON-ы дополнительно обрабатываются заранее, и полученные значения кладутся в отдельные столбцы в уже обработанном виде.) Все эти функции исходят из сильных допущений о том, каким может быть JSON, и при этом стараются почти ничего не делать.

Делаются следующие допущения:

1. Имя поля (аргумент функции) должно быть константой;
2. Считается, что имя поля в JSON-е закодировано некоторым каноническим образом. Например, `visitParamHas('{"abc":"def"}', 'abc') = 1`, но `visitParamHas('{"\u0061\u0062\u0063":"def"}', 'abc') = 0`
3. Поля ищутся на любом уровне вложенности, без разбора. Если есть несколько подходящих полей - берётся первое.
4. В JSON-е нет пробельных символов вне строковых литералов.

### visitParamHas(params, name)

Проверяет наличие поля с именем `name`.

Синоним: `simpleJSONHas`.

### visitParamExtractUInt(params, name)

Пытается выделить число типа UInt64 из значения поля с именем `name`. Если поле строковое, пытается выделить число из начала строки. Если такого поля нет, или если оно есть, но содержит не число, то возвращает 0.

Синоним: `simpleJSONExtractUInt`.

### visitParamExtractInt(params, name)

Аналогично для Int64.

Синоним: `simpleJSONExtractInt`.

### visitParamExtractFloat(params, name)

Аналогично для Float64.

Синоним: `simpleJSONExtractFloat`.

### visitParamExtractBool(params, name)

Пытается выделить значение true/false. Результат — UInt8.

Синоним: `simpleJSONExtractBool`.

## visitParamExtractRaw(params, name)

Возвращает значение поля, включая разделители.

Синоним: `simpleJSONExtractRaw`.

Примеры:

```
visitParamExtractRaw('{"abc":"\\n\\u0000"}', 'abc') = '\"\\n\\u0000\"';  
visitParamExtractRaw('{"abc":{"def":[1,2,3]}}', 'abc') = '{"def":[1,2,3]}';
```

## visitParamExtractString(params, name)

Разбирает строку в двойных кавычках. У значения убирается экранирование. Если убрать экранированные символы не удалось, то возвращается пустая строка.

Синоним: `simpleJSONExtractString`.

Примеры:

```
visitParamExtractString('{"abc":"\\n\\u0000"}', 'abc') = '\\n\\0';  
visitParamExtractString('{"abc":"\\u263a"}', 'abc') = '@';  
visitParamExtractString('{"abc":"\\u263"}', 'abc') = '';  
visitParamExtractString('{"abc":"hello"}', 'abc') = '';
```

На данный момент не поддерживаются записанные в формате `\uXXXX\uYYYY` кодовые точки не из basic multilingual plane (они переводятся не в UTF-8, а в CESU-8).

Следующие функции используют [simdjson](#), который разработан под более сложные требования для разбора JSON. Упомянутое выше допущение 2 по-прежнему применимо.

## isValidJSON(json)

Проверяет, является ли переданная строка валидным json значением.

Примеры:

```
SELECT isValidJSON('{"a": "hello", "b": [-100, 200.0, 300]}') = 1  
SELECT isValidJSON('not a json') = 0
```

## JSONHas(json[, indices\_or\_keys]...)

Если значение существует в документе JSON, то возвращается 1.

Если значение не существует, то возвращается 0.

Примеры:

```
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` — это список из нуля или более аргументов каждый из них может быть либо строкой либо целым числом.

- Стока — это доступ к объекту по ключу.

- Положительное целое число — это доступ к n-му члену/ключу с начала.
- Отрицательное целое число — это доступ к n-му члену/ключу с конца.

Адресация элементов по индексу начинается с 1, следовательно элемент 0 не существует.

Вы можете использовать целые числа, чтобы адресовать как массивы JSON, так и JSON-объекты.

Примеры:

```
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]...)

Возвращает длину массива JSON или объекта JSON.

Если значение не существует или имеет неверный тип, то возвращается 0.

Примеры:

```
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]...)

Возвращает тип значения JSON.

Если значение не существует, то возвращается Null.

Примеры:

```
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]...)

Парсит JSON и извлекает значение. Эти функции аналогичны функциям visitParam.

Если значение не существует или имеет неверный тип, то возвращается 0.

Примеры:

```
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]...)

Парсит JSON и извлекает строку. Эта функция аналогична функции visitParamExtractString.

Если значение не существует или имеет неверный тип, то возвращается пустая строка.

У значения убирается экранирование. Если убрать экранированные символы не удалось, то возвращается пустая строка.

Примеры:

```
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)

Парсит JSON и извлекает значение с заданным типом данных.

Это обобщение предыдущих функций `JSONExtract<type>`.

Это означает

`JSONExtract(..., 'String')` выдает такой же результат, как `JSONExtractString()`,  
`JSONExtract(..., 'Float64')` выдает такой же результат, как `JSONExtractFloat()`.

Примеры:

```
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(\u2019Sunday\u2019 = 0, \u2019Monday\u2019 = 1, \u2019Tuesday\u2019 = 2,  
\u2019Wednesday\u2019 = 3, \u2019Thursday\u2019 = 4, \u2019Friday\u2019 = 5, \u2019Saturday\u2019 = 6)') = 'Thursday'  
SELECT JSONExtract('{"day": 5}', 'day', 'Enum8(\u2019Sunday\u2019 = 0, \u2019Monday\u2019 = 1, \u2019Tuesday\u2019 = 2, \u2019Wednesday\u2019 = 3,  
\u2019Thursday\u2019 = 4, \u2019Friday\u2019 = 5, \u2019Saturday\u2019 = 6)') = 'Friday'
```

## JSONExtractKeysAndValues(json[, indices\_or\_keys...], Value\_type)

Разбор пар ключ-значение из JSON, где значение имеет тип данных ClickHouse.

Пример:

```
SELECT JSONExtractKeysAndValues('{"x": {"a": 5, "b": 7, "c": 11}}', 'x', 'Int8') = [('a',5),('b',7),('c',11)];
```

## JSONExtractRaw(json[, indices\_or\_keys]...)

Возвращает часть JSON в виде строки, содержащей неразобранный подстроку.

Если значение не существует, то возвращается пустая строка.

Пример:

```
SELECT JSONExtractRaw('{"a": "hello", "b": [-100, 200.0, 300]}', 'b') = '[-100, 200.0, 300]';
```

## JSONExtractArrayRaw(json[, indices\_or\_keys]...)

Возвращает массив из элементов JSON массива, каждый из которых представлен в виде строки с неразобранными подстроками из JSON.

Если значение не существует или не является массивом, то возвращается пустой массив.

Пример:

```
SELECT JSONExtractArrayRaw('{"a": "hello", "b": [-100, 200.0, "hello"]}', 'b') = ['-100', '200.0', "hello"];
```

## JSONExtractKeysAndValuesRaw

Извлекает необработанные данные из объекта JSON.

### Синтаксис

```
JSONExtractKeysAndValuesRaw(json[, p, a, t, h])
```

### Аргументы

- **json** — строка, содержащая валидный JSON.
- **p, a, t, h** — индексы или ключи, разделенные запятыми, которые указывают путь к внутреннему полю во вложенном объекте JSON. Каждый аргумент может быть либо строкой для получения поля по ключу, либо целым числом для получения N-го поля (индексирование начинается с 1, отрицательные числа используются для отсчета с конца). Если параметр не задан, весь JSON парсится как объект верхнего уровня. Необязательный параметр.

### Возвращаемые значения

- Массив с кортежами ('key', 'value'). Члены кортежа — строки.
- Пустой массив, если заданный объект не существует или входные данные не валидный JSON.

Тип: **Array(Tuple(String, String))**.

### Примеры

Запрос:

```
SELECT JSONExtractKeysAndValuesRaw('{"a": [-100, 200.0], "b": {"c": {"d": "hello", "f": "world"} }}');
```

Результат:

```
JSONExtractKeysAndValuesRaw('{"a": [-100, 200.0], "b": {"c": {"d": "hello", "f": "world"} }}')-->[('a',[-100,200]),('b',{"c":{"d":"hello","f":"world"} })]
```

Запрос:

```
SELECT JSONExtractKeysAndValuesRaw('{"a": [-100, 200.0], "b": {"c": {"d": "hello", "f": "world"} }}', 'b');
```

Результат:

```
JSONExtractKeysAndValuesRaw('{"a": [-100, 200.0], "b":{"c": {"d": "hello", "f": "world"}}}', 'b')  
[('c','{"d":"hello","f":"world"})]
```

Запрос:

```
SELECT JSONExtractKeysAndValuesRaw('{"a": [-100, 200.0], "b":{"c": {"d": "hello", "f": "world"}}}', -1, 'c');
```

Результат:

```
JSONExtractKeysAndValuesRaw('{"a": [-100, 200.0], "b":{"c": {"d": "hello", "f": "world"}}}', -1, 'c')  
[('d',"hello"),('f',"world")]
```

## JSON\_EXISTS(json, path)

Если значение существует в документе JSON, то возвращается 1.

Если значение не существует, то возвращается 0.

Пример:

```
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]');
```

### Примечание

до версии 21.11 порядок аргументов функции был обратный, т.е. JSON\_EXISTS(path, json)

## JSON\_QUERY(json, path)

Парсит JSON и извлекает значение как JSON массив или JSON объект.

Если значение не существует, то возвращается пустая строка.

Пример:

```
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'));
```

Результат:

```
["world"]  
[0, 1, 4, 0, -1, -4]  
[2]  
String
```

## Примечание

до версии 21.11 порядок аргументов функции был обратный, т.е. `JSON_QUERY(path, json)`

## JSON\_VALUE(json, path)

Парсит JSON и извлекает значение как JSON скаляр.

Если значение не существует, то возвращается пустая строка.

Пример:

```
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'));
```

Результат:

```
"world"
0
2
String
```

## Примечание

до версии 21.11 порядок аргументов функции был обратный, т.е. `JSON_VALUE(path, json)`

## toJSONString

Сериализует значение в JSON представление. Поддерживаются различные типы данных и вложенные структуры.

По умолчанию 64-битные [целые числа](#) и более (например, `UInt64` или `Int128`) заключаются в кавычки.

Настройка `output_format_json_quote_64bit_integers` управляет этим поведением.

Специальные значения `Nan` и `inf` заменяются на `null`. Чтобы они отображались, включите настройку `output_format_json_quote_denormals`.

Когда сериализуется значение [Enum](#), то функция выводит его имя.

### Синтаксис

```
toJSONString(value)
```

### Аргументы

- `value` — значение, которое необходимо сериализовать. Может быть любого типа.

### Возвращаемое значение

- JSON представление значения.

Тип: [String](#).

### Пример

Первый пример показывает сериализацию [Map](#).

Во втором примере есть специальные значения, обернутые в [Tuple](#).

Запрос:

```
SELECT toJSONString(map('key1', 1, 'key2', 2));
SELECT toJSONString(tuple(1.25, NULL, NaN, +inf, -inf, [])) SETTINGS output_format_json_quote_denormals = 1;
```

Результат:

```
{"key1":1,"key2":2}
[1.25,null,"nan","inf",-inf,[]]
```

## Смотрите также

- [output\\_format\\_json\\_quote\\_64bit\\_integers](#)
- [output\\_format\\_json\\_quote\\_denormals](#)

## Внимание

Для словарей, созданных с помощью [DDL-запросов](#), в параметре `dict_name` указывается полное имя словаря вместе с базой данных, например: `<database>.dict_name`. Если база данных не указана, используется текущая.

# ФУНКЦИИ ДЛЯ РАБОТЫ С ВНЕШНИМИ СЛОВАРЯМИ

Информацию о подключении и настройке внешних словарей смотрите в разделе [Внешние словари](#).

## dictGet, dictGetOrDefault, dictGetOrNull

Извлекает значение из внешнего словаря.

```
dictGet('dict_name', attr_names, id_expr)
dictGetOrDefault('dict_name', attr_names, id_expr, default_value_expr)
dictGetOrNull('dict_name', attr_name, id_expr)
```

## Аргументы

- `dict_name` — имя словаря. [Строковый литерал](#).
- `attr_names` — имя столбца словаря. [Строковый литерал](#), или кортеж [Tuple](#) таких имен.
- `id_expr` — значение ключа словаря. [Expression](#) возвращает пару "ключ-значение" словаря или [Tuple](#), в зависимости от конфигурации словаря.
- `default_value_expr` — значение, возвращаемое в том случае, когда словарь не содержит строки с заданным ключом `id_expr`. [Выражение](#), возвращающее значение с типом данных, сконфигурированным для атрибута `attr_names`, или кортеж [Tuple](#) таких выражений.

## Возвращаемое значение

- Значение атрибута, соответствующее ключу `id_expr`, если ClickHouse смог привести это значение к [заданному типу данных](#).

- Если ключа, соответствующего `id_expr` в словаре нет, то:
  - `dictGet` возвращает содержимое элемента `<null_value>`, указанного для атрибута в конфигурации словаря.
  - `dictGetOrDefault` возвращает атрибут `default_value_expr`.
  - `dictGetOrNull` возвращает `NULL` в случае, если ключ не найден в словаре.

Если значение атрибута не удалось обработать или оно не соответствует типу данных атрибута, то ClickHouse генерирует исключение.

### Пример с единственным атрибутом

Создадим текстовый файл `ext-dict-text.csv` со следующим содержимым:

```
1,1
2,2
```

Первый столбец — `id`, второй столбец — `c1`.

Настройка внешнего словаря:

```
<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>
```

Выполним запрос:

```
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

### Пример с несколькими атрибутами

Создадим текстовый файл ext-dict-mult.csv со следующим содержимым:

```
1,1,'1'  
2,2,'2'  
3,3,'3'
```

Первый столбец — id, второй столбец — c1, третий столбец — c2.

Настройка внешнего словаря:

```
<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>
```

Выполним запрос:

```
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)

### Пример для словаря с диапазоном ключей

Создадим таблицу:

```

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');

```

Создадим внешний словарь:

```

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);

```

Выполним запрос:

```

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;

```

Результат:

(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)

## Смотрите также

- [Внешние словари](#)

## dictHas

Проверяет, присутствует ли запись с указанным ключом в словаре.

```
dictHas('dict_name', id)
```

## Аргументы

- `dict_name` — имя словаря. [Строковый литерал](#).

- `id_expr` — значение ключа словаря. [Expression](#) возвращает пару "ключ-значение" словаря или [Tuple](#) в зависимости от конфигурации словаря.

## Возвращаемое значение

- 0, если ключа нет.
- 1, если ключ есть.

Тип: [UInt8](#).

## dictGetHierarchy

Создаёт массив, содержащий цепочку предков для заданного ключа в [иерархическом словаре](#).

### Синтаксис

```
dictGetHierarchy('dict_name', key)
```

### Аргументы

- `dict_name` — имя словаря. [Строковый литерал](#).
- `key` — значение ключа. [Выражение](#), возвращающее значение типа [UInt64](#).

## Возвращаемое значение

- Цепочка предков заданного ключа.

Тип: [Array\(UInt64\)](#).

## dictIsIn

Проверяет предка ключа по всей иерархической цепочке словаря.

```
dictIsIn ('dict_name', child_id_expr, ancestor_id_expr)
```

### Аргументы

- `dict_name` — имя словаря. [Строковый литерал](#).
- `child_id_expr` — ключ для проверки. [Выражение](#), возвращающее значение типа [UInt64](#).
- `ancestor_id_expr` — предполагаемый предок ключа `child_id_expr`. [Выражение](#), возвращающее значение типа [UInt64](#).

## Возвращаемое значение

- 0, если `child_id_expr` — не дочерний элемент `ancestor_id_expr`.
- 1, если `child_id_expr` — дочерний элемент `ancestor_id_expr` или если `child_id_expr` есть `ancestor_id_expr`.

Тип: [UInt8](#).

## dictGetChildren

Возвращает потомков первого уровня в виде массива индексов. Это обратное преобразование для [dictGetHierarchy](#).

### Синтаксис

```
dictGetChildren(dict_name, key)
```

## Аргументы

- `dict_name` — имя словаря. [String literal](#).
- `key` — значение ключа. [Выражение](#), возвращающее значение типа [UInt64](#).

## Возвращаемые значения

- Потомки первого уровня для ключа.

Тип: [Array\(UInt64\)](#).

## Пример

Рассмотрим иерархический словарь:

id	parent_id
1	0
2	1
3	1
4	2

Потомки первого уровня:

```
SELECT dictGetChildren('hierarchy_flat_dictionary', number) FROM system.numbers LIMIT 4;
```

```
dictGetChildren('hierarchy_flat_dictionary', number)─  
[1] ──────────  
[2,3] ──────────  
[4] ──────────  
[] ──────────
```

## dictGetDescendant

Возвращает всех потомков, как если бы функция [dictGetChildren](#) была выполнена `level` раз рекурсивно.

## Синтаксис

```
dictGetDescendants(dict_name, key, level)
```

## Аргументы

- `dict_name` — имя словаря. [String literal](#).
- `key` — значение ключа. [Выражение](#), возвращающее значение типа [UInt64](#).
- `level` — уровень иерархии. Если `level = 0`, возвращаются все потомки. [UInt8](#).

## Возвращаемые значения

- Потомки для ключа.

Тип: [Array\(UInt64\)](#).

## Пример

Рассмотрим иерархический словарь:

id	parent_id
1	0
2	1
3	1
4	2

Все потомки:

```
SELECT dictGetDescendants('hierarchy_flat_dictionary', number) FROM system.numbers LIMIT 4;
```

dictGetDescendants('hierarchy_flat_dictionary', number)
[1,2,3,4]
[2,3,4]
[4]
[]

Потомки первого уровня:

```
SELECT dictGetDescendants('hierarchy_flat_dictionary', number, 1) FROM system.numbers LIMIT 4;
```

dictGetDescendants('hierarchy_flat_dictionary', number, 1)
[1]
[2,3]
[4]
[]

## Прочие функции

ClickHouse поддерживает специализированные функции, которые приводят значения атрибутов словаря к определённому типу данных независимо от конфигурации словаря.

Функции:

- `dictGetInt8`, `dictGetInt16`, `dictGetInt32`, `dictGetInt64`
- `dictGetUInt8`, `dictGetUInt16`, `dictGetUInt32`, `dictGetUInt64`
- `dictGetFloat32`, `dictGetFloat64`
- `dictGetDate`
- `dictGetDateTime`
- `dictGetUUID`
- `dictGetString`

Все эти функции можно использовать с модификатором `OrDefault`. Например, `dictGetDateOrDefault`.

Синтаксис:

```
dictGet[Type](#sql-reference-functions--dict_name---attr_name---id_expr)
dictGet[TypeOrDefault('dict_name', 'attr_name', id_expr, default_value_expr)
```

## Аргументы

- `dict_name` — имя словаря. [Строковый литерал](#).
- `attr_name` — имя столбца словаря. [Строковый литерал](#).
- `id_expr` — значение ключа словаря. [Выражение](#), возвращающее значение типа [UInt64](#) или [Tuple](#) в зависимости от конфигурации словаря.
- `default_value_expr` — значение, возвращаемое в том случае, когда словарь не содержит строки с заданным ключом `id_expr`. [Выражение](#), возвращающее значение с типом данных, сконфигурированным для атрибута `attr_name`.

## Возвращаемое значение

- Если ClickHouse успешно обработал атрибут в соответствии с [заданным типом данных](#), то функции возвращают значение атрибута, соответствующее ключу `id_expr`.
- Если запрошенного `id_expr` нет в словаре, то:
  - `dictGet[Type]` возвращает содержимое элемента `<null_value>`, указанного для атрибута в конфигурации словаря.
  - `dictGet[TypeOrDefault]` возвращает аргумент `default_value_expr`.

Если значение атрибута не удалось обработать или оно не соответствует типу данных атрибута, то ClickHouse генерирует исключение.

## ФУНКЦИИ ДЛЯ РАБОТЫ СО СЛОВАРЯМИ Яндекс.Метрики

Чтобы указанные ниже функции работали, в конфиге сервера должны быть указаны пути и адреса для получения всех словарей Яндекс.Метрики. Словари загружаются при первом вызове любой из этих функций. Если справочники не удаётся загрузить - будет выкинуто исключение.

О том, как создать справочники, смотрите в разделе «Словари».

## Множественные геобазы

ClickHouse поддерживает работу одновременно с несколькими альтернативными геобазами (иерархиями регионов), для того чтобы можно было поддержать разные точки зрения о принадлежности регионов странам.

В конфиге `clickhouse-server` указывается файл с иерархией регионов:

```
<path_to_regions_hierarchy_file>/opt/geo/regions_hierarchy.txt</path_to_regions_hierarchy_file>
```

Кроме указанного файла, рядом ищутся файлы, к имени которых (до расширения) добавлен символ `_` и какой угодно суффикс.

Например, также найдётся файл `/opt/geo/regions_hierarchy_ua.txt`, если такой есть.

`ua` называется ключом словаря. Для словаря без суффикса, ключ является пустой строкой.

Все словари перезагружаются в рантайме (раз в количество секунд, заданное в конфигурационном параметре `builtin_dictionaries_reload_interval`, по умолчанию - раз в час), но перечень доступных словарей определяется один раз, при старте сервера.

Все функции по работе с регионами, в конце добавлен один необязательный аргумент - ключ словаря. Далее он обозначен как `qeobase`.

Пример:

`regionToCountry(RegionID)` - использует словарь по умолчанию: `/opt/geo/regions_hierarchy.txt`;  
`regionToCountry(RegionID, "")` - использует словарь по умолчанию: `/opt/geo/regions_hierarchy.txt`;  
`regionToCountry(RegionID, 'ua')` - использует словарь для ключа ua: `/opt/geo/regions_hierarchy_ua.txt`;

`regionToCity(id[, geobase])`

Принимает число типа UInt32 - идентификатор региона из геобазы Яндекса. Если регион является городом или входит в некоторый город, то возвращает идентификатор региона - соответствующего города. Иначе возвращает 0.

**regionToArea(id[, geobase])**

Переводит регион в область (тип в геобазе - 5). В остальном, аналогично функции regionToCity.

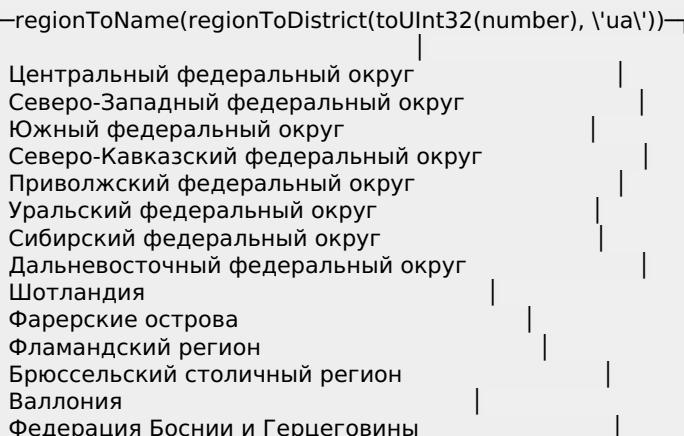
```
SELECT DISTINCT regionToName(regionToArea(toUInt32(number), 'ua'))
FROM system.numbers
LIMIT 15
```

```
—regionToName(regionToArea(toUInt32(number), '\ua1'))—  
|  
Москва и Московская область |  
Санкт-Петербург и Ленинградская область |  
Белгородская область |  
Ивановская область |  
Калужская область |  
Костромская область |  
Курская область |  
Липецкая область |  
Орловская область |  
Рязанская область |  
Смоленская область |  
Тамбовская область |  
Тверская область |  
Тульская область |
```

`regionToDistrict(id[, geobase])`

Переводит регион в федеральный округ (тип в геобазе - 4). В остальном, аналогично функции regionToCity.

```
SELECT DISTINCT regionToName(regionToDistrict(toUInt32(number), 'ua'))
FROM system.numbers
LIMIT 15
```



## regionToCountry(id[, geobase])

Переводит регион в страну. В остальном, аналогично функции `regionToCity`.

Пример: `regionToCountry(toUInt32(213)) = 225` - преобразовали Москву (213) в Россию (225).

## regionToContinent(id[, geobase])

Переводит регион в континент. В остальном, аналогично функции `regionToCity`.

Пример: `regionToContinent(toUInt32(213)) = 10001` - преобразовали Москву (213) в Евразию (10001).

## regionToTopContinent (#regiontotopcontinent)

Находит для региона верхний в иерархии континент.

### Синтаксис

```
regionToTopContinent(id[, geobase])
```

### Аргументы

- `id` — идентификатор региона из геобазы Яндекса. [UInt32](#).
- `geobase` — ключ словаря. Смотрите [Множественные геобазы](#). [String](#). Опциональный параметр.

### Возвращаемое значение

- Идентификатор континента верхнего уровня (последний при подъеме по иерархии регионов).
- 0, если его нет.

Тип: [UInt32](#).

## regionToPopulation(id[, geobase])

Получает население для региона.

Население может быть прописано в файлах с геобазой. Смотрите в разделе «[Встроенные словари](#)». Если для региона не прописано население, возвращается 0.

В геобазе Яндекса, население может быть прописано для дочерних регионов, но не прописано для родительских.

## regionIn(lhs, rhs[, geobase])

Проверяет принадлежность региона lhs региону rhs. Возвращает число типа UInt8, равное 1, если принадлежит и 0, если не принадлежит.

Отношение рефлексивное - любой регион принадлежит также самому себе.

## regionHierarchy(id[, geobase])

Принимает число типа UInt32 - идентификатор региона из геобазы Яндекса. Возвращает массив идентификаторов регионов, состоящий из переданного региона и всех родителей по цепочке.

Пример: `regionHierarchy(toUInt32(213)) = [213,1,3,225,10001,10000]`.

## regionToName(id[, lang])

Принимает число типа UInt32 - идентификатор региона из геобазы Яндекса. Вторым аргументом может быть передана строка - название языка. Поддерживаются языки ru, en, ua, uk, by, kz, tr. Если второй аргумент отсутствует - используется язык ru. Если язык не поддерживается - кидается исключение. Возвращает строку - название региона на соответствующем языке. Если региона с указанным идентификатором не существует - возвращается пустая строка.

ua и uk обозначают одно и то же - украинский язык.

## Функции для реализации оператора IN

### in, notIn, globalIn, globalNotIn

Смотрите раздел [Операторы IN](#).

## Функция ArrayJoin

Это совсем необычная функция.

Обычные функции не изменяют множество строк, а лишь изменяют значения в каждой строке (map).

Агрегатные функции выполняют свёртку множества строк (fold, reduce).

Функция arrayJoin выполняет размножение каждой строки в множество строк (unfold).

Функция принимает в качестве аргумента массив, и размножает исходную строку в несколько строк - по числу элементов массива.

Все значения в столбцах просто копируются, кроме значения в столбце с применением этой функции - он заменяется на соответствующее значение массива.

В запросе может быть использовано несколько функций `arrayJoin`. В этом случае, соответствующее преобразование делается несколько раз.

Обратите внимание на синтаксис ARRAY JOIN в запросе SELECT, который предоставляет более широкие возможности.

Пример:

```
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]

# Функции для работы с географическими координатами

## greatCircleDistance

Вычисляет расстояние между двумя точками на поверхности Земли по [формуле большого круга](#).

```
greatCircleDistance(lon1Deg, lat1Deg, lon2Deg, lat2Deg)
```

### Входные параметры

- `lon1Deg` — долгота первой точки в градусах. Диапазон —  $[-180^\circ, 180^\circ]$ .
- `lat1Deg` — широта первой точки в градусах. Диапазон —  $[-90^\circ, 90^\circ]$ .
- `lon2Deg` — долгота второй точки в градусах. Диапазон —  $[-180^\circ, 180^\circ]$ .
- `lat2Deg` — широта второй точки в градусах. Диапазон —  $[-90^\circ, 90^\circ]$ .

Положительные значения соответствуют северной широте и восточной долготе, отрицательные — южной широте и западной долготе.

### Возвращаемое значение

Расстояние между двумя точками на поверхности Земли в метрах.

Генерирует исключение, когда значения входных параметров выходят за границы диапазонов.

### Пример

```
SELECT greatCircleDistance(55.755831, 37.617673, -55.755831, -37.617673)
```

```
greatCircleDistance(55.755831, 37.617673, -55.755831, -37.617673) —  
14132374.194975413 |
```

## greatCircleAngle

Вычисляет угловое расстояние на сфере по [формуле большого круга](#).

```
greatCircleAngle(lon1Deg, lat1Deg, lon2Deg, lat2Deg)
```

### Входные параметры

- `lon1Deg` — долгота первой точки в градусах.
- `lat1Deg` — широта первой точки в градусах.
- `lon2Deg` — долгота второй точки в градусах.
- `lat2Deg` — широта второй точки в градусах.

### Возвращаемое значение

Длина дуги большого круга между двумя точками в градусах.

## Пример

```
SELECT greatCircleAngle(0, 0, 45, 0) AS arc
```

```
arc  
45
```

## pointInEllipses

Проверяет, принадлежит ли точка хотя бы одному из эллипсов.

Координаты — геометрические в декартовой системе координат.

```
pointInEllipses(x, y, x0, y0, a0, b0, ..., xn, yn, an, bn)
```

### Входные параметры

- $x, y$  — координаты точки на плоскости.
- $x_i, y_i$  — координаты центра  $i$ -го эллипса.
- $a_i, b_i$  — полуоси  $i$ -го эллипса (в единицах измерения координат  $x, y$ ).

Входных параметров должно быть  $2+4\cdot n$ , где  $n$  — количество эллипсов.

### Возвращаемые значения

1, если точка внутри хотя бы одного из эллипсов, 0, если нет.

## Пример

```
SELECT pointInEllipses(10., 10., 10., 9.1, 1., 0.9999)
```

```
pointInEllipses(10., 10., 10., 9.1, 1., 0.9999)  
1
```

## pointInPolygon

Проверяет, принадлежит ли точка многоугольнику на плоскости.

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

### Входные значения

- $(x, y)$  — координаты точки на плоскости. Тип данных — **Tuple** — кортеж из двух чисел.
- $[(a, b), (c, d) ...]$  — вершины многоугольника. Тип данных — **Array**. Каждая вершина представлена парой координат  $(a, b)$ . Вершины следует указывать в порядке обхода по или против часовой стрелки. Минимальное количество вершин — 3. Многоугольник должен быть константным.
- функция поддерживает также многоугольники с дырками (вырезанными кусками). Для этого случая, добавьте многоугольники, описывающие вырезанные куски, дополнительными аргументами функции. Функция не поддерживает не одно связные многоугольники.

### Возвращаемые значения

1, если точка внутри многоугольника, 0, если нет.

Если точка находится на границе многоугольника, функция может возвращать как 0, так и 1.

### Пример

```
SELECT pointInPolygon((3., 3.), [(6, 0), (8, 4), (5, 8), (0, 2)]) AS res
```

```
res  
1 |
```

## ФУНКЦИИ ДЛЯ РАБОТЫ С ИНДЕКСАМИ Н3

**Н3** — это система геокодирования, которая делит поверхность Земли на равные шестиугольные ячейки. Система поддерживает иерархию (вложенность) ячеек, т.е. каждый "родительский" шестиугольник может быть поделен на семь одинаковых вложенных "дочерних" шестиугольников, и так далее.

Уровень вложенности называется "разрешением" и может принимать значение от 0 до 15, где 0 соответствует "базовым" ячейкам самого верхнего уровня (наиболее крупным).

Для каждой точки, имеющей широту и долготу, можно получить 64-битный индекс Н3, соответствующий номеру шестиугольной ячейки, где эта точка находится.

Индексы Н3 используются, в основном, для геопозиционирования и расчета расстояний.

### h3IsValid

Проверяет корректность **Н3**-индекса.

#### Синтаксис

```
h3IsValid(h3index)
```

#### Параметр

- **h3index** — идентификатор шестиугольника. Тип данных: **UInt64**.

#### Возвращаемые значения

- 1 — число является Н3-индексом.
- 0 — число не является Н3-индексом.

Тип: **UInt8**.

### Пример

Запрос:

```
SELECT h3IsValid(630814730351855103) AS h3IsValid;
```

Результат:

```
h3IsValid
  1 |
```

## h3GetResolution

Извлекает разрешение H3-индекса.

### Синтаксис

```
h3GetResolution(h3index)
```

### Параметр

- h3index — идентификатор шестигранника. Тип данных: [UInt64](#).

### Возвращаемые значения

- Разрешение сетки. Диапазон значений: [0, 15].
- Для несуществующего идентификатора может быть возвращено произвольное значение. Используйте [h3IsValid](#) для проверки идентификаторов.

Тип: [UInt8](#).

### Пример

Запрос:

```
SELECT h3GetResolution(639821929606596015) AS resolution;
```

Результат:

```
resolution
  14 |
```

## h3EdgeAngle

Рассчитывает средний размер стороны шестигранника H3 в градусах.

### Синтаксис

```
h3EdgeAngle(resolution)
```

### Параметр

- resolution — требуемое разрешение индекса. Тип данных: [UInt8](#). Диапазон возможных значений: [0, 15].

### Возвращаемое значение

- Средняя длина стороны шестигранника H3 в градусах. Тип данных: [Float64](#).

### Пример

Запрос:

```
SELECT h3EdgeAngle(10) AS edgeAngle;
```

Результат:

```
h3EdgeAngle(10)  
0.0005927224846720883 |
```

## h3EdgeLengthM

Рассчитывает средний размер стороны шестиугранника **H3** в метрах.

### Синтаксис

```
h3EdgeLengthM(resolution)
```

### Параметр

- `resolution` — требуемое разрешение индекса. Тип данных — **UInt8**. Диапазон возможных значений — [0, 15].

### Возвращаемое значение

- Средняя длина стороны шестиугранника **H3** в метрах, тип — **Float64**.

### Пример

Запрос:

```
SELECT h3EdgeLengthM(15) AS edgeLengthM;
```

Результат:

```
edgeLengthM  
0.509713273 |
```

## geoToH3

Возвращает **H3** индекс точки (`lon`, `lat`) с заданным разрешением.

### Синтаксис

```
geoToH3(lon, lat, resolution)
```

### Аргументы

- `lon` — географическая долгота. Тип данных — **Float64**.
- `lat` — географическая широта. Тип данных — **Float64**.
- `resolution` — требуемое разрешение индекса. Тип данных — **UInt8**. Диапазон возможных значений — [0, 15].

### Возвращаемые значения

- Порядковый номер шестигранника.
- 0 в случае ошибки.

Тип данных: **UInt64**.

## Пример

Запрос:

```
SELECT geoToH3(37.79506683, 55.71290588, 15) AS h3Index;
```

Результат:

```
h3Index  
644325524701193974
```

## h3ToGeo

Возвращает географические координаты долготы и широты, соответствующие указанному **H3**-индексу.

### Синтаксис

```
h3ToGeo(h3Index)
```

### Аргументы

- **h3Index** — **H3**-индекс. **UInt64**.

### Возвращаемые значения

- кортеж из двух значений: `tuple(lon,lat)`, где `lon` — долгота **Float64**, `lat` — широта **Float64**.

## Пример

Запрос:

```
SELECT h3ToGeo(644325524701193974) coordinates;
```

Результат:

```
coordinates  
(37.79506616830252,55.71290243145668)
```

## h3kRing

Возвращает **H3**-индексы шестигранников в радиусе `k` от данного в произвольном порядке.

### Синтаксис

```
h3kRing(h3index, k)
```

### Аргументы

- `h3index` — идентификатор шестигранника. Тип данных: [UInt64](#).
- `k` — радиус. Тип данных: [целое число](#)

### Возвращаемые значения

- Массив из НЗ-индексов.

Тип данных: [Array\(UInt64\)](#).

### Пример

Запрос:

```
SELECT arrayJoin(h3kRing(644325529233966508, 1)) AS h3index;
```

Результат:

```
h3index
644325529233966508
644325529233966497
644325529233966510
644325529233966504
644325529233966509
644325529233966355
644325529233966354
```

## h3GetBaseCell

Определяет номер базовой (верхнеуровневой) шестиугольной [НЗ](#)-ячейки для указанной ячейки.

### Синтаксис

```
h3GetBaseCell(index)
```

### Параметр

- `index` — индекс шестиугольной ячейки. Тип: [UInt64](#).

### Возвращаемое значение

- Индекс базовой шестиугольной ячейки.

Тип: [UInt8](#).

### Пример

Запрос:

```
SELECT h3GetBaseCell(612916788725809151) AS basecell;
```

Результат:

```
basecell
12 |
```

## h3HexAreaM2

Определяет среднюю площадь шестиугольной **H3**-ячейки заданного разрешения в квадратных метрах.

## Синтаксис

```
h3HexAreaM2(resolution)
```

## Параметр

- `resolution` — разрешение. Диапазон: [0, 15]. Тип: **UInt8**.

## Возвращаемое значение

- Площадь в квадратных метрах. Тип: **Float64**.

## Пример

Запрос:

```
SELECT h3HexAreaM2(13) AS area;
```

Результат:

```
area
43.9 |
```

## h3IndexesAreNeighbors

Определяет, являются ли **H3**-ячейки соседями.

## Синтаксис

```
h3IndexesAreNeighbors(index1, index2)
```

## Аргументы

- `index1` — индекс шестиугольной ячейки. Тип: **UInt64**.
- `index2` — индекс шестиугольной ячейки. Тип: **UInt64**.

## Возвращаемое значение

- `1` — ячейки являются соседями.
- `0` — ячейки не являются соседями.

Тип: **UInt8**.

## Пример

Запрос:

```
SELECT h3IndexesAreNeighbors(617420388351344639, 617420388352655359) AS n;
```

Результат:

1

## h3ToChildren

Формирует массив дочерних (вложенных) НЗ-ячеек для указанной ячейки.

### Синтаксис

```
h3ToChildren(index, resolution)
```

### Аргументы

- `index` — индекс шестиугольной ячейки. Тип: `UInt64`.
- `resolution` — разрешение. Диапазон: [0, 15]. Тип: `UInt8`.

### Возвращаемое значение

- Массив дочерних НЗ-ячеек.

Тип: `Array(UInt64)`.

### Пример

Запрос:

```
SELECT h3ToChildren(599405990164561919, 6) AS children;
```

Результат:

```
└─children
   └─[603909588852408319, 603909588986626047, 603909589120843775, 603909589255061503, 603909589389279231,
      603909589523496959, 603909589657714687] |
```

## h3ToParent

Определяет родительскую (более крупную) НЗ-ячейку, содержащую указанную ячейку.

### Синтаксис

```
h3ToParent(index, resolution)
```

### Аргументы

- `index` — индекс шестиугольной ячейки. Тип: `UInt64`.
- `resolution` — разрешение. Диапазон: [0, 15]. Тип: `UInt8`.

### Возвращаемое значение

- Индекс родительской НЗ-ячейки.

Тип: [UInt64](#).

## Пример

Запрос:

```
SELECT h3ToParent(599405990164561919, 3) AS parent;
```

Результат:

```
parent
590398848891879423
```

## h3ToString

Преобразует [H3](#)-индекс из числового представления [H3Index](#) в строковое.

```
h3ToString(index)
```

## Параметр

- `index` — индекс шестиугольной ячейки. Тип: [UInt64](#).

## Возвращаемое значение

- Строковое представление H3-индекса.

Тип: [String](#).

## Пример

Запрос:

```
SELECT h3ToString(617420388352917503) AS h3_string;
```

Результат:

```
h3_string
89184926cdbfffff |
```

## stringToH3

Преобразует [H3](#)-индекс из строкового представления в числовое представление [H3Index](#).

## Синтаксис

```
stringToH3(index_str)
```

## Параметр

- `index_str` — строковое представление H3-индекса. Тип: [String](#).

## Возвращаемое значение

- Числовое представление индекса шестиугольной ячейки.

- 0, если при преобразовании возникла ошибка.

Тип: **UInt64**.

### Пример

Запрос:

```
SELECT stringToH3('89184926cc3ffff') AS index;
```

Результат:

```
index  
617420388351344639
```

## h3GetResolution

Определяет разрешение H3-ячейки.

### Синтаксис

```
h3GetResolution(index)
```

### Параметр

- `index` — индекс шестиугольной ячейки. Тип: **UInt64**.

### Возвращаемое значение

- Разрешение ячейки. Диапазон: [0, 15].

Тип: **UInt8**.

### Пример

Запрос:

```
SELECT h3GetResolution(617420388352917503) AS res;
```

Результат:

```
res  
9 |
```

## h3IsResClassIII

Проверяет, имеет ли индекс H3 разрешение с ориентацией Class III.

### Синтаксис

```
h3IsResClassIII(index)
```

### Параметр

- `index` — порядковый номер шестигранника. Тип: **UInt64**.

## Возвращаемые значения

- 1 — индекс имеет разрешение с ориентацией Class III.
- 0 — индекс не имеет разрешения с ориентацией Class III.

Тип: [UInt8](#).

## Пример

Запрос:

```
SELECT h3IsResClassIII(617420388352917503) AS res;
```

Результат:

```
res
1 |
```

## h3IsPentagon

Проверяет, является ли указанный индекс [H3](#) пятиугольной ячейкой.

## Синтаксис

```
h3IsPentagon(index)
```

## Параметр

- `index` — порядковый номер шестигранника. Тип: [UInt64](#).

## Возвращаемые значения

- 1 — индекс представляет собой пятиугольную ячейку.
- 0 — индекс не является пятиугольной ячейкой.

Тип: [UInt8](#).

## Пример

Запрос:

```
SELECT h3IsPentagon(644721767722457330) AS pentagon;
```

Результат:

```
pentagon
0 |
```

## h3GetFaces

Возвращает все грани многоугольника (икосаэдра), пересекаемые заданным [H3](#) индексом.

## Синтаксис

```
h3GetFaces(index)
```

## Параметр

- index — индекс шестиугольной ячейки. Тип: **UInt64**.

## Возвращаемое значение

- Массив, содержащий грани многоугольника (икосаэдра), пересекаемые заданным НЗ индексом.

Тип: **Array(UInt64)**.

## Пример

Запрос:

```
SELECT h3GetFaces(599686042433355775) AS faces;
```

Результат:

```
faces  
[7] |
```

toc\_title: "Функции для работы с индексами S2"

# ФУНКЦИИ ДЛЯ РАБОТЫ С СИСТЕМОЙ Geohash

**Geohash** — это система геокодирования, которая делит поверхность Земли на участки в виде "решетки", и каждую ячейку решетки кодирует в виде строки из букв и цифр. Система поддерживает иерархию (вложенность) ячеек, поэтому чем точнее определена геопозиция, тем длиннее строка с кодом соответствующей ячейки.

Для ручного преобразования географических координат в строку geohash можно использовать сайт [geohash.org](http://geohash.org).

## geohashEncode

Кодирует широту и долготу в строку **geohash**.

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

## Входные значения

- longitude — долгота. Диапазон — [-180°, 180°].
- latitude — широта. Диапазон — [-90°, 90°].
- precision — длина результирующей строки, по умолчанию 12. Опционально. Целое число в диапазоне [1, 12]. Любое значение меньше, чем 1 или больше 12 автоматически преобразуются в 12.

## Возвращаемые значения

- Стока с координатой, закодированной модифицированной версией алфавита base32.

## Пример

```
SELECT geohashEncode(-5.60302734375, 42.593994140625, 0) AS res;
```

```
res  
ezs42d000000 |
```

## geohashDecode

Декодирует любую строку, закодированную в **geohash**, на долготу и широту.

```
geohashDecode(geohash_string)
```

### Входные значения

- `geohash_string` — строка, содержащая geohash.

### Возвращаемые значения

- `(longitude, latitude)` — широта и долгота. Кортеж из двух значений типа `Float64`.

## Пример

```
SELECT geohashDecode('ezs42') AS res;
```

```
res  
(-5.60302734375,42.60498046875) |
```

## geohashesInBox

Формирует массив участков, которые находятся внутри или пересекают границу заданного участка на поверхности. Каждый участок описывается строкой **geohash** заданной точности.

### Синтаксис

```
geohashesInBox(longitude_min, latitude_min, longitude_max, latitude_max, precision)
```

### Аргументы

- `longitude_min` — минимальная долгота. Диапазон возможных значений:  $[-180^\circ, 180^\circ]$ . Тип данных: **Float**.
- `latitude_min` — минимальная широта. Диапазон возможных значений:  $[-90^\circ, 90^\circ]$ . Тип данных: **Float**.
- `longitude_max` — максимальная долгота. Диапазон возможных значений:  $[-180^\circ, 180^\circ]$ . Тип данных: **Float**.
- `latitude_max` — максимальная широта. Диапазон возможных значений:  $[-90^\circ, 90^\circ]$ . Тип данных: **Float**.
- `precision` — точность geohash. Диапазон возможных значений:  $[1, 12]$ . Тип данных: **UInt8**.

## Замечание

Все передаваемые координаты должны быть одного и того же типа: либо `Float32`, либо `Float64`.

### Возвращаемые значения

- Массив строк, описывающих участки, покрывающие заданный участок. Длина каждой строки соответствует точности geohash. Порядок строк — произвольный.
- [ ] - Если переданные минимальные значения широты и долготы больше соответствующих максимальных значений, функция возвращает пустой массив.

Тип данных: `Array(String)`.

## Замечание

Если возвращаемый массив содержит свыше 10 000 000 элементов, функция сгенерирует исключение.

### Пример

Запрос:

```
SELECT geohashesInBox(24.48, 40.56, 24.785, 40.81, 4) AS thasos;
```

Результат:

```
thasos
['sx1q','sx1r','sx32','sx1w','sx1x','sx38'] |
```

## Функции для работы с Nullable-аргументами

### isNull

Проверяет является ли аргумент `NUL`.

```
isNull(x)
```

Синоним: `ISNULL`.

### Аргументы

- `x` — значение с не составным типом данных.

### Возвращаемое значение

- 1, если `x` — `NUL`.
- 0, если `x` — не `NUL`.

### Пример

## Входная таблица

x	y
1	NULL
2	3

## Запрос

```
SELECT x FROM t_null WHEREisNull(y);
```

x
1

## isNotNull

Проверяет не является ли аргумент **NULL**.

```
isNotNull(x)
```

## Аргументы

- `x` — значение с не составным типом данных.

## Возвращаемое значение

- 0, если `x` — NULL.
- 1, если `x` — не NULL.

## Пример

### Входная таблица

x	y
1	NULL
2	3

## Запрос

```
SELECT x FROM t_null WHERE isNotNull(y);
```

x
2

## coalesce

Последовательно слева-направо проверяет являются ли переданные аргументы **NULL** и возвращает первый не **NULL**.

```
coalesce(x,...)
```

## Аргументы

- Произвольное количество параметров не составного типа. Все параметры должны быть совместимы по типу данных.

## Возвращаемые значения

- Первый не `NULL` аргумент.
- `NULL`, если все аргументы — `NULL`.

## Пример

Рассмотрим адресную книгу, в которой может быть указано несколько способов связи с клиентом.

name	mail	phone	icq
client 1	NULL	123-45-67	123
client 2	NULL	NULL	NULL

Поля `mail` и `phone` имеют тип `String`, а поле `icq` — `UInt32`, его необходимо будет преобразовать в `String`.

Получим из адресной книги первый доступный способ связаться с клиентом:

```
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

Возвращает альтернативное значение, если основной аргумент — `NULL`.

```
ifNull(x,alt)
```

## Аргументы

- `x` — значение для проверки на `NULL`,
- `alt` — значение, которое функция вернёт, если `x` — `NULL`.

## Возвращаемые значения

- Значение `x`, если `x` — не `NULL`.
- Значение `alt`, если `x` — `NULL`.

## Пример

```
SELECT ifNull('a', 'b');
```

ifNull('a', 'b')
a

```
SELECT ifNull(NULL, 'b');
```

```
ifNull(NULL, 'b')  
b
```

## nullIf

Возвращает `NULL`, если аргументы равны.

```
nullIf(x, y)
```

### Аргументы

`x, y` — значения для сравнения. Они должны быть совместимых типов, иначе ClickHouse сгенерирует исключение.

### Возвращаемые значения

- `NULL`, если аргументы равны.
- Значение `x`, если аргументы не равны.

### Пример

```
SELECT nullIf(1, 1);
```

```
nullIf(1, 1)  
NULL
```

```
SELECT nullIf(1, 2);
```

```
nullIf(1, 2)  
1
```

## assumeNotNull

Приводит значение типа `Nullable` к не `Nullable`, если значение не `NULL`.

```
assumeNotNull(x)
```

### Аргументы

- `x` — исходное значение.

### Возвращаемые значения

- Исходное значение с не `Nullable` типом, если оно — не `NULL`.
- Неспецифицированный результат, зависящий от реализации, если исходное значение — `NULL`.

### Пример

Рассмотрим таблицу `t_null`.

```
SHOW CREATE TABLE t_null;
```

statement  
CREATE TABLE default.t\_null ( x Int8, y Nullable(Int8) ) ENGINE = TinyLog |

x	y
1	NULL
2	3

Применим функцию `assumeNotNull` к столбцу `y`.

```
SELECT assumeNotNull(y) FROM t_null;
```

assumeNotNull(y)  
0 |  
3 |

```
SELECT toTypeName(assumeNotNull(y)) FROM t_null;
```

toTypeName(assumeNotNull(y))  
Int8 |  
Int8 |

## toNullable

Преобразует тип аргумента к `Nullable`.

```
toNullable(x)
```

### Аргументы

- `x` — значение произвольного не составного типа.

### Возвращаемое значение

- Входное значение с типом не `Nullable`.

### Пример

```
SELECT toTypeName(10);
```

toTypeName(10)  
UInt8 |

```
SELECT toTypeName(toNullable(10));
```

```
toTypeName(toNullable(10))—  
 Nullable(UInt8) |
```

## ФУНКЦИИ МАШИННОГО ОБУЧЕНИЯ

### evalMLMethod (prediction)

Предсказание с использованием подобранных регрессионных моделей.

#### Stochastic Linear Regression

Агрегатная функция [stochasticLinearRegression](#) реализует стохастический градиентный спуск, используя линейную модель и функцию потерь MSE.

#### Stochastic Logistic Regression

Агрегатная функция [stochasticLogisticRegression](#) реализует стохастический градиентный спуск для задачи бинарной классификации.

## ФУНКЦИИ ИНТРОСПЕКЦИИ

Функции из этого раздела могут использоваться для интроспекции [ELF](#) и [DWARF](#) в целях профилирования запросов.

### Предупреждение

Эти функции выполняются медленно и могут приводить к нежелательным последствиям в плане безопасности.

Для правильной работы функций интроспекции:

- Установите пакет `clickhouse-common-static-dbg`.
- Установите настройку `allow_introspection_functions` в 1.

Из соображений безопасности данные функции отключены по умолчанию.

ClickHouse сохраняет отчеты профилировщика в [журнал трассировки](#) в системной таблице. Убедитесь, что таблица и профилировщик настроены правильно.

## addresssToLine

Преобразует адрес виртуальной памяти внутри процесса сервера ClickHouse в имя файла и номер строки в исходном коде ClickHouse.

Если вы используете официальные пакеты ClickHouse, вам необходимо установить следующий пакеты: `clickhouse-common-static-dbg`.

### Синтаксис

```
addressToLine(address_of_binary_instruction)
```

## Аргументы

- address\_of\_binary\_instruction ([Тип UInt64](#))- Адрес инструкции в запущенном процессе.

## Возвращаемое значение

- Имя файла исходного кода и номер строки в этом файле разделяются двоеточием.

Например, `/build/obj-x86\_64-linux-gnu/../src/Common/ThreadPool.cpp:199`, где `199` — номер строки.

- Имя бинарного файла, если функция не может найти отладочную информацию.
- Пустая строка, если адрес не является допустимым.

Тип: [String](#).

## Пример

Включение функций самоанализа:

```
SET allow_introspection_functions=1;
```

Выбор первой строки из списка `trace_log` системная таблица:

```
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:	[140658411141617,94784174532828,94784076370703,94784076372094,94784076361020,94784175007680,140658411116251,140658403895439]

To `trace` поле содержит трассировку стека в момент выборки.

Получение имени файла исходного кода и номера строки для одного адреса:

```
SELECT addressToLine(94784076370703) \G;
```

Row 1:

```
addressToLine(94784076370703): /build/obj-x86_64-linux-gnu/../src/Common/ThreadPool.cpp:199
```

Применение функции ко всему стектрейсу:

```
SELECT
    arrayStringConcat(arrayMap(x -> addressToLine(x), trace), '\n') AS trace_source_code_lines
FROM system.trace_log
LIMIT 1
\G
```

Функция `arrayMap` позволяет обрабатывать каждый отдельный элемент массива `trace` с помощью функции `addressToLine`. Результат этой обработки вы видите в виде `trace_source_code_lines` колонки выходных данных.

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

Преобразует адрес виртуальной памяти внутри серверного процесса ClickHouse в символ из объектных файлов ClickHouse.

### Синтаксис

```
addressToSymbol(address_of_binary_instruction)
```

### Аргументы

- `address_of_binary_instruction` ([Тип uint64](#)) — адрес инструкции в запущенном процессе.

### Возвращаемое значение

- Символ из объектных файлов ClickHouse.
- Пустая строка, если адрес не является допустимым.

Тип: [String](#).

### Пример

Включение функций самоанализа:

```
SET allow_introspection_functions=1;
```

Выбор первой строки из списка `trace_log` системная таблица:

```
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:
[94138803686098,94138815010911,94138815096522,9413881501224,94138815102091,94138814222988,94138806823642,94138814457211,94138806823642,94138814457211,94138806823642,94138806795179,94138806796,94138753770094,94138753771646,94138753760572,94138852407232,140399185266395,140399178045583]
```

To trace поле содержит трассировку стека в момент выборки.

Получение символа для одного адреса:

```
SELECT addressToSymbol(94138803686098) \G;
```

Row 1:

```
addressToSymbol(94138803686098):
_ZNK2DB24IAggregateFunctionHelperINS_20AggregateFunctionSumImmNS_24AggregateFunctionSumDataImEEEEEE1
9addBatchSinglePlaceEmPcPPKNS_7IColumnEPNS_5ArenaE
```

Применение функции ко всей трассировке стека:

```
SELECT
  arrayStringConcat(arrayMap(x -> addressToSymbol(x), trace), '\n') AS trace_symbols
FROM system.trace_log
LIMIT 1
\G
```

То **arrayMap** функция позволяет обрабатывать каждый отдельный элемент системы. trace массив по типу **addressToSymbols** функция. Результат этой обработки вы видите в виде **trace\_symbols** колонка выходных данных.

Row 1:

```
trace_symbols:
_ZNK2DB24IAggregateFunctionHelperINS_20AggregateFunctionSumImmNS_24AggregateFunctionSumDataImEEEEEE1
9addBatchSinglePlaceEmPcPPKNS_7IColumnEPNS_5ArenaE
_ZN2DB10Aggregator21executeWithoutKeyImplERPcmPNS0_28AggregateFunctionInstructionEPNS_5ArenaE
_ZN2DB10Aggregator14executeOnBlockESt6vectorIN3COWINS_7IColumnEE13immutable_ptrIS3_EESaIS6_EEmRNS_2
2AggregatedDataVariantsERS1_IPKS3_SaISC_EERS1_ISE_SalSE_EERb
_ZN2DB10Aggregator14executeOnBlockERKNS_5BlockERNS_22AggregatedDataVariantsERSt6vectorIPKNS_7IColumn
ESaIS9_EERS6_ISB_SalSB_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_E12scheduleImplvEET_St8functionIFvvEEiSt8optionalImEEU
vE1_JEEEOS4_DpOT0_ENKUIvE_cIEv
_ZN14ThreadPoolImplISt6threadE6workerESt14_List_iteratorIS0_E
execute_native_thread_routine
start_thread
clone
```

# demangle

Преобразует символ, который вы можете получить с помощью [addressstosymbol](#) функция имя функции C++.

## Синтаксис

```
demangle(symbol)
```

## Аргументы

- **symbol** ([Строка](#)) - символ из объектного файла.

## Возвращаемое значение

- Имя функции C++.
- Пустая строка, если символ не является допустимым.

Тип: [Строка](#).

## Пример

Включение функций самоанализа:

```
SET allow_introspection_functions=1;
```

Выбор первой строки из списка `trace_log` системная таблица:

```
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:
[94138803686098,94138815010911,94138815096522,94138815101224,94138815102091,94138814222988,9413806823642,94138814457211,94138806823642,94138814457211,94138806823642,94138806795179,94138806796144,94138753770094,94138753771646,94138753760572,94138852407232,140399185266395,140399178045583]
```

To `trace` поле содержит трассировку стека в момент выборки.

Получение имени функции для одного адреса:

```
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
```

Применение функции ко всему стектрейсу:

```
SELECT
    arrayStringConcat(arrayMap(x -> demangle(addressToSymbol(x)), trace), '\n') AS trace_functions
FROM system.trace_log
LIMIT 1
\G
```

Функция `arrayMap` позволяет обрабатывать каждый отдельный элемент массива `trace` с помощью функции `demangle`.

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<void ()> DB::AsynchronousBlockInputStream::next()::
{lambda()#1}::_M_invoke(std::any 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

Возвращает id потока, в котором обрабатывается текущий `Block`.

### Синтаксис

```
tid()
```

### Возвращаемое значение

- Id текущего потока. `UInt64`.

### Пример

Запрос:

```
SELECT tid();
```

Результат:

```
tid()  
3878 |
```

## logTrace

Выводит сообщение в лог сервера для каждого Block.

### Синтаксис

```
logTrace('message')
```

### Аргументы

- `message` — сообщение, которое отправляется в серверный лог. [String](#).

### Возвращаемое значение

- Всегда возвращает 0.

### Пример

Запрос:

```
SELECT logTrace('logTrace message');
```

Результат:

```
logTrace('logTrace message')  
0 |
```

## Прочие функции

### hostName()

Возвращает строку - имя хоста, на котором эта функция была выполнена. При распределённой обработке запроса, это будет имя хоста удалённого сервера, если функция выполняется на удалённом сервере.

Если функция вызывается в контексте распределенной таблицы, то она генерирует обычный столбец со значениями, актуальными для каждого шарда. Иначе возвращается константа.

### getMacro

Возвращает именованное значение из секции [macros](#) конфигурации сервера.

### Синтаксис

```
getMacro(name)
```

### Аргументы

- `name` — имя, которое необходимо получить из секции [macros](#). [String](#).

### Возвращаемое значение

- Значение по указанному имени.

Тип: **String**.

### Пример

Пример секции `macros` в конфигурационном файле сервера:

```
<macros>
  <test>Value</test>
</macros>
```

Запрос:

```
SELECT getMacro('test');
```

Результат:

getMacro('test')	
Value	

Альтернативный способ получения значения:

```
SELECT * FROM system.macros
WHERE macro = 'test'
```

macro	substitution
test	Value

## FQDN

Возвращает полное имя домена.

### Синтаксис

```
fqdn()
```

Эта функция регистронезависимая.

### Возвращаемое значение

- Полное имя домена.

Тип: **String**.

### Пример

Запрос:

```
SELECT FQDN();
```

Результат:

```
FQDN()  
clickhouse.ru-central1.internal |
```

## basename

Извлекает конечную часть строки после последнего слэша или бэкслэша. Функция часто используется для извлечения имени файла из пути.

```
basename( expr )
```

### Аргументы

- `expr` — выражение, возвращающее значение типа **String**. В результирующем значении все бэкслэши должны быть экранированы.

### Возвращаемое значение

Строка, содержащая:

- Конечную часть строки после последнего слэша или бэкслэша.

Если входная строка содержит путь, заканчивающийся слэшем или бэкслэшем, например, `/` или `c:\`, функция возвращает пустую строку.

- Исходная строка, если нет слэша или бэкслэша.

### Пример

```
SELECT 'some/long/path/to/file' AS a, basename(a);
```

```
a      basename('some\\long\\path\\to\\file')  
some\\long\\path\\to\\file | file |
```

```
SELECT 'some\\long\\path\\to\\file' AS a, basename(a);
```

```
a      basename('some\\long\\path\\to\\file')  
some\\long\\path\\to\\file | file |
```

```
SELECT 'some-file-name' AS a, basename(a);
```

```
a      basename('some-file-name')  
some-file-name | some-file-name |
```

## visibleWidth(x)

Вычисляет приблизительную ширину при выводе значения в текстовом (tab-separated) виде на консоль.

Функция используется системой для реализации Pretty форматов.

`NULL` представляется как строка, соответствующая отображению `NULL` в форматах Pretty.

```
SELECT visibleWidth(NULL)
```

```
visibleWidth(NULL) └  
  4 |
```

## toTypeName(x)

Возвращает строку, содержащую имя типа переданного аргумента.

Если на вход функции передать `NULL`, то она вернёт тип `Nullable(Nothing)`, что соответствует внутреннему представлению `NULL` в ClickHouse.

## blockSize()

Получить размер блока.

В ClickHouse выполнение запроса всегда идёт по блокам (наборам кусочков столбцов). Функция позволяет получить размер блока, для которого её вызвали.

## byteSize

Возвращает оценку в байтах размера аргументов в памяти в несжатом виде.

### Синтаксис

```
byteSize(argument [, ...])
```

### Аргументы

- `argument` — значение.

### Возвращаемое значение

- Оценка размера аргументов в памяти в байтах.

Тип: `UInt64`.

### Примеры

Для аргументов типа `String` функция возвращает длину строки + 9 (нуль-терминатор + длина)

Запрос:

```
SELECT byteSize('string');
```

Результат:

```
byteSize('string') └  
  15 |
```

Запрос:

```

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;

```

Результат:

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

```

Если функция принимает несколько аргументов, то она возвращает их совокупный размер в байтах.

Запрос:

```
SELECT byteSize(NULL, 1, 0.3, "");
```

Результат:

byteSize(NULL, 1, 0.3, "")	19
----------------------------	----

## materialize(x)

Превращает константу в полноценный столбец, содержащий только одно значение.

В ClickHouse полноценные столбцы и константы представлены в памяти по-разному. Функции по-разному работают для аргументов-констант и обычных аргументов (выполняется разный код), хотя результат почти всегда должен быть одинаковым. Эта функция предназначена для отладки такого поведения.

## ignore(...)

Принимает любые аргументы, в т.ч. `NULL`, всегда возвращает 0.

При этом, аргумент всё равно вычисляется. Это может использоваться для бенчмарков.

## sleep(seconds)

Спит `seconds` секунд на каждый блок данных. Можно указать как целое число, так и число с плавающей запятой.

## currentDatabase()

Возвращает имя текущей базы данных.

Эта функция может использоваться в параметрах движка таблицы в запросе `CREATE TABLE` там, где нужно указать базу данных.

## currentUser()

Возвращает логин текущего пользователя. При распределенном запросе, возвращается имя пользователя, инициировавшего запрос.

```
SELECT currentUser();
```

Алиас: `user()`, `USER()`.

### Возвращаемые значения

- Логин текущего пользователя.
- При распределенном запросе — логин пользователя, инициировавшего запрос.

Тип: `String`.

### Пример

Запрос:

```
SELECT currentUser();
```

Результат:

```
currentUser()  
default |
```

## isConstant

Проверяет, является ли аргумент константным выражением.

Константное выражение — это выражение, результат которого известен на момент анализа запроса (до его выполнения). Например, выражения над [литералами](#) являются константными.

Используется в целях разработки, отладки или демонстрирования.

### Синтаксис

```
isConstant(x)
```

### Аргументы

- $x$  — выражение для проверки.

## Возвращаемые значения

- 1 — выражение  $x$  является константным.
- 0 — выражение  $x$  не является константным.

Тип: **UInt8**.

## Примеры

Запрос:

```
SELECT isConstant(x + 1) FROM (SELECT 43 AS x);
```

Результат:

```
isConstant(plus(x, 1))  
 1 |
```

Запрос:

```
WITH 3.14 AS pi SELECT isConstant(cos(pi));
```

Результат:

```
isConstant(cos(pi))  
 1 |
```

Запрос:

```
SELECT isConstant(number) FROM numbers(1)
```

Результат:

```
isConstant(number)  
 0 |
```

## isFinite( $x$ )

Принимает `Float32` или `Float64` и возвращает `UInt8`, равный 1, если аргумент не бесконечный и не `NaN`, иначе 0.

## ifNotFinite

Проверяет, является ли значение дробного числа с плавающей точкой конечным.

### Синтаксис

```
ifNotFinite(x,y)
```

## Аргументы

- `x` — значение, которое нужно проверить на бесконечность. Тип: `Float*`.
- `y` — запасное значение. Тип: `Float*`.

## Возвращаемые значения

- `x`, если `x` принимает конечное значение.
- `y`, если `x` принимает не конечное значение.

## Пример

Запрос:

```
SELECT 1/0 as infimum, ifNotFinite(infimum,42)
```

Результат:

```
infimum--ifNotFinite(divide(1, 0), 42)--  
inf | 42 |
```

Аналогичный результат можно получить с помощью [тернарного оператора](#) `isFinite(x) ? x : y`.

## isInfinite(x)

Принимает `Float32` или `Float64` и возвращает `UInt8`, равный 1, если аргумент бесконечный, иначе 0. Отметим, что в случае `NAN` возвращается 0.

## isNaN(x)

Принимает `Float32` или `Float64` и возвращает `UInt8`, равный 1, если аргумент является `NAN`, иначе 0.

## hasColumnInTable(['hostname'][, 'username'][, 'password'][],] 'database', 'table', 'column')

Принимает константные строки - имя базы данных, имя таблицы и название столбца. Возвращает константное выражение типа `UInt8`, равное 1,

если есть столбец, иначе 0. Если задан параметр `hostname`, проверка будет выполнена на удалённом сервере.

Функция кидает исключение, если таблица не существует.

Для элементов вложенной структуры данных функция проверяет существование столбца. Для самой же вложенной структуры данных функция возвращает 0.

## bar

Позволяет построить unicode-art диаграмму.

`bar(x, min, max, width)` рисует полосу ширины пропорциональной (`x - min`) и равной `width` символов при `x = max`.

Аргументы:

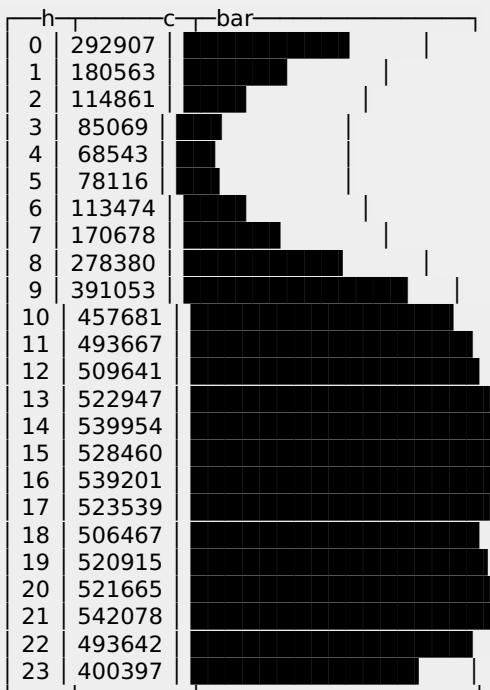
- `x` — Величина для отображения.
- `min, max` — Целочисленные константы, значение должно помещаться в `Int64`.

- `width` — Константа, положительное число, может быть дробным.

Полоса рисуется с точностью до одной восьмой символа.

Пример:

```
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

Преобразовать значение согласно явно указанному отображению одних элементов на другие.  
Имеется два варианта функции:

`transform(x, array_from, array_to, default)`

`x` - что преобразовывать.

`array_from` - константный массив значений для преобразования.

`array_to` - константный массив значений, в которые должны быть преобразованы значения из `from`.

`default` - какое значение использовать, если `x` не равен ни одному из значений во `from`.

`array_from` и `array_to` - массивы одинаковых размеров.

Типы:

`transform(T, Array(T), Array(U), U) -> U`

**T** и **U** - могут быть числовыми, строковыми, или Date или DateTime типами.

При этом, где обозначена одна и та же буква (T или U), могут быть, в случае числовых типов, не совпадающие типы, а типы, для которых есть общий тип.

Например, первый аргумент может иметь тип Int64, а второй - Array(UInt16).

Если значение x равно одному из элементов массива array\_from, то возвращает соответствующий (такой же по номеру) элемент массива array\_to; иначе возвращает default. Если имеется несколько совпадающих элементов в array\_from, то возвращает какой-нибудь из соответствующих.

Пример:

```
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)

Отличается от первого варианта отсутствующим аргументом default.

Если значение x равно одному из элементов массива array\_from, то возвращает соответствующий (такой же по номеру) элемент массива array\_to; иначе возвращает x.

Типы:

```
transform(T, Array(T), Array(T)) -> T
```

Пример:

```
SELECT
    transform(domain(Referer), ['yandex.ru', 'google.ru', 'vk.com'], ['www.yandex', 'example.com']) AS s,
    count() AS c
FROM test.hits
GROUP BY domain(Referer)
ORDER BY count() DESC
LIMIT 10
```

s	c
www.yandex	2906259
[REDACTED].ru	867767
mail.yandex.ru	313599
[REDACTED].ru	107147
news.yandex.ru	100355
[REDACTED].ru	65040
example.com	64515
[REDACTED].net	59141
example.com	57316

## formatReadableSize(x)

Принимает размер (число байт). Возвращает округленный размер с суффиксом (KiB, MiB и т.д.) в виде строки.

Пример:

```
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)

Принимает число. Возвращает округленное число с суффиксом (thousand, million, billion и т.д.) в виде строки.

Облегчает визуальное восприятие больших чисел живым человеком.

Пример:

```
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

## least(a, b)

Возвращает наименьшее значение из a и b.

## greatest(a, b)

Возвращает наибольшее значение из a и b.

## uptime()

Возвращает аптайм сервера в секундах.

Если функция вызывается в контексте распределенной таблицы, то она генерирует обычный столбец со значениями, актуальными для каждого шарда. Иначе возвращается константа.

## version()

Возвращает версию сервера в виде строки.

Если функция вызывается в контексте распределенной таблицы, то она генерирует обычный столбец со значениями, актуальными для каждого шарда. Иначе возвращается константа.

## buildId()

Возвращает ID сборки, сгенерированный компилятором для данного сервера ClickHouse.

Если функция вызывается в контексте распределенной таблицы, то она генерирует обычный столбец со значениями, актуальными для каждого шарда. Иначе возвращается константа.

## rowNumberInBlock

Возвращает порядковый номер строки в блоке данных. Для каждого блока данных нумерация начинается с 0.

## rowNumberInAllBlocks()

Возвращает порядковый номер строки в блоке данных. Функция учитывает только задействованные блоки данных.

## neighbor

Функция позволяет получить доступ к значению в столбце `column`, находящемуся на смещении `offset` относительно текущей строки. Является частичной реализацией [оконных функций LEAD\(\) и LAG\(\)](#).

### Синтаксис

```
neighbor(column, offset[, default_value])
```

Результат функции зависит от затронутых блоков данных и порядка данных в блоке.

### Предупреждение

Функция может получить доступ к значению в столбце соседней строки только внутри обрабатываемого в данный момент блока данных.

Порядок строк, используемый при вычислении функции `neighbor`, может отличаться от порядка строк, возвращаемых пользователю.

Чтобы этого не случилось, вы можете сделать подзапрос с `ORDER BY` и вызвать функцию извне подзапроса.

### Аргументы

- `column` — имя столбца или скалярное выражение.
- `offset` — смещение от текущей строки `column`. [Int64](#).
- `default_value` — опциональный параметр. Значение, которое будет возвращено, если смещение выходит за пределы блока данных.

### Возвращаемое значение

- Значение `column` в смещении от текущей строки, если значение `offset` не выходит за пределы блока.
- Значение по умолчанию для `column`, если значение `offset` выходит за пределы блока данных. Если передан параметр `default_value`, то значение берется из него.

Тип: зависит от данных в `column` или переданного значения по умолчанию в `default_value`.

### Пример

Запрос:

```
SELECT number, neighbor(number, 2) FROM system.numbers LIMIT 10;
```

Результат:

number	neighbor(number, 2)
0	2
1	3
2	4
3	5
4	6
5	7
6	8
7	9
8	0
9	0

Запрос:

```
SELECT number, neighbor(number, 2, 999) FROM system.numbers LIMIT 10;
```

Результат:

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

Эта функция может использоваться для оценки year-over-year значение показателя:

Запрос:

```
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)
```

Результат:

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)

Считает разницу между последовательными значениями строк в блоке данных.

Возвращает 0 для первой строки и разницу с предыдущей строкой для каждой последующей строки.

### Предупреждение

Функция может взять значение предыдущей строки только внутри текущего обработанного блока данных.

Результат функции зависит от затронутых блоков данных и порядка данных в блоке.

Порядок строк, используемый при вычислении функции `runningDifference`, может отличаться от порядка строк, возвращаемых пользователю.

Чтобы этого не случилось, вы можете сделать подзапрос с `ORDER BY` и вызвать функцию извне подзапроса.

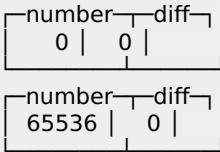
Пример:

```
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

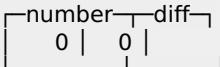
Обратите внимание — размер блока влияет на результат. С каждым новым блоком состояние `runningDifference` сбрасывается.

```
SELECT
    number,
    runningDifference(number + 1) AS diff
FROM numbers(100000)
WHERE diff != 1
```



```
set max_block_size=100000 -- по умолчанию 65536!
```

```
SELECT
    number,
    runningDifference(number + 1) AS diff
FROM numbers(100000)
WHERE diff != 1
```



## runningDifferenceStartingWithFirstValue

То же, что и `runningDifference`, но в первой строке возвращается значение первой строки, а не ноль.

## runningConcurrency

Подсчитывает количество одновременно идущих событий.

У каждого события есть время начала и время окончания. Считается, что время начала включено в событие, а время окончания исключено из него. Столбцы со временем начала и окончания событий должны иметь одинаковый тип данных.

Функция подсчитывает количество событий, происходящих одновременно на момент начала каждого из событий в выборке.

### Предупреждение

События должны быть отсортированы по возрастанию времени начала. Если это требование нарушено, то функция вызывает исключение.

Каждый блок данных обрабатывается независимо. Если события из разных блоков данных накладываются по времени, они не могут быть корректно обработаны.

### Синтаксис

```
runningConcurrency(start, end)
```

### Аргументы

- `start` — Столбец с временем начала событий. `Date`, `DateTime` или `DateTime64`.

- `end` — Столбец с временем окончания событий. `Date`, `DateTime` или `DateTime64`.

## Возвращаемое значение

- Количество одновременно идущих событий на момент начала каждого события.

Тип: `UInt32`

## Пример

Рассмотрим таблицу:

	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	

Запрос:

```
SELECT start, runningConcurrency(start, end) FROM example_table;
```

Результат:

	start	runningConcurrency(start, end)
2021-03-03		1
2021-03-06		2
2021-03-07		3
2021-03-11		2

## MACNumToString(num)

Принимает число типа `UInt64`. Интерпретирует его, как MAC-адрес в big endian. Возвращает строку, содержащую соответствующий MAC-адрес в формате AA:BB:CC:DD:EE:FF (числа в шестнадцатеричной форме через двоеточие).

## MACStringToNum(s)

Функция, обратная к `MACNumToString`. Если MAC адрес в неправильном формате, то возвращает 0.

## MACStringToOUI(s)

Принимает MAC адрес в формате AA:BB:CC:DD:EE:FF (числа в шестнадцатеричной форме через двоеточие). Возвращает первые три октета как число в формате `UInt64`. Если MAC адрес в неправильном формате, то возвращает 0.

## getSizeOfEnumType

Возвращает количество полей в `Enum`.

```
getSizeOfEnumType(value)
```

## Аргументы

- `value` — значение типа `Enum`.

## Возвращаемые значения

- Количество полей входного значения типа `Enum`.
- Исключение, если тип не `Enum`.

## Пример

```
SELECT getSizeOfEnumType( CAST('a' AS Enum8('a' = 1, 'b' = 2) ) ) AS x
```

```
x  
2 |
```

## blockSerializedSize

Возвращает размер на диске (без учета сжатия).

```
blockSerializedSize(value[, value[, ...]])
```

## Аргументы

- `value` — значение произвольного типа.

## Возвращаемые значения

- Количество байтов, которые будут записаны на диск для блока значений (без сжатия).

## Пример

Запрос:

```
SELECT blockSerializedSize(maxState(1)) as x
```

Ответ:

```
x  
2 |
```

## toColumnName

Возвращает имя класса, которым представлен тип данных столбца в оперативной памяти.

```
toColumnName(value)
```

## Аргументы

- `value` — значение произвольного типа.

## Возвращаемые значения

- Стока с именем класса, который используется для представления типа данных `value` в оперативной памяти.

## Пример разницы между `toTypeName` и `toColumnName`

```
SELECT toTypeName(CAST('2018-01-01 01:02:03' AS DateTime))
```

```
toTypeName(CAST('2018-01-01 01:02:03', 'DateTime'))  
|  
DateTime
```

```
SELECT toColumnTypeName(CAST('2018-01-01 01:02:03' AS DateTime))
```

```
toColumnTypeName(CAST('2018-01-01 01:02:03', 'DateTime'))  
|  
Const(UInt32)
```

В примере видно, что тип данных `DateTime` хранится в памяти как `Const(UInt32)`.

## dumpColumnStructure

Выводит развернутое описание структур данных в оперативной памяти

```
dumpColumnStructure(value)
```

### Аргументы

- `value` — значение произвольного типа.

### Возвращаемые значения

- Стока с описанием структуры, которая используется для представления типа данных `value` в оперативной памяти.

### Пример

```
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))
```

## defaultValueOfTypeArgumentType

Выводит значение по умолчанию для типа данных.

Не учитывает значения по умолчанию для столбцов, заданные пользователем.

```
defaultValueOfTypeArgumentType(expression)
```

### Аргументы

- `expression` — значение произвольного типа или выражение, результатом которого является значение произвольного типа.

### Возвращаемые значения

- 0 для чисел;

- Пустая строка для строк;
- `NULL` для **Nullable**.

## Пример

```
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

Выводит значение по умолчанию для указанного типа данных.

Не включает значения по умолчанию для настраиваемых столбцов, установленных пользователем.

```
defaultValueOfTypeName(type)
```

## Аргументы

- `type` — тип данных.

## Возвращаемое значение

- 0 для чисел;
- Пустая строка для строк;
- `NULL` для **Nullable**.

## Пример

```
SELECT defaultValueOfTypeName('Int8')
```

```
defaultValueOfTypeName('Int8')—  
0 |
```

```
SELECT defaultValueOfTypeName('Nullable(Int8)')
```

```
defaultValueOfTypeName('Nullable(Int8)')—  
NULL |
```

## indexHint

Возвращает все данные из диапазона, в который попадают данные, соответствующие указанному выражению.

Переданное выражение не будет вычислено. Выбор диапазона производится по индексу.

Индекс в ClickHouse разреженный, при чтении диапазона в ответ попадают «лишние» соседние данные.

## Синтаксис

```
SELECT * FROM table WHERE indexHint(<expression>)
```

## Возвращаемое значение

Возвращает диапазон индекса, в котором выполняется заданное условие.

Тип: [UInt8](#).

## Пример

Рассмотрим пример с использованием тестовых данных таблицы [ontime](#).

Исходная таблица:

```
SELECT count() FROM ontime
```

```
count()  
4276457
```

В таблице есть индексы по полям (`FlightDate`, (`Year`, `FlightDate`)).

Выполним выборку по дате, где индекс не используется.

Запрос:

```
SELECT FlightDate AS k, count() FROM ontime GROUP BY k ORDER BY k
```

ClickHouse обработал всю таблицу (Processed 4.28 million rows).

Результат:

k	count()
2017-01-01	13970
2017-01-02	15882
.....	.....
2017-09-28	16411
2017-09-29	16384
2017-09-30	12520

Для подключения индекса выбираем конкретную дату.

Запрос:

```
SELECT FlightDate AS k, count() FROM ontime WHERE k = '2017-09-15' GROUP BY k ORDER BY k
```

При использовании индекса ClickHouse обработал значительно меньшее количество строк (Processed 32.74 thousand rows).

Результат:

k	count()
2017-09-15	16428

Передадим в функцию indexHint выражение k = '2017-09-15'.

Запрос:

```
SELECT
    FlightDate AS k,
    count()
FROM ontime
WHERE indexHint(k = '2017-09-15')
GROUP BY k
ORDER BY k ASC
```

ClickHouse применил индекс по аналогии с примером выше (Processed 32.74 thousand rows).

Выражение k = '2017-09-15' не используется при формировании результата.

Функция indexHint позволяет увидеть соседние данные.

Результат:

k	count()
2017-09-14	7071
2017-09-15	16428
2017-09-16	1077
2017-09-30	8167

## replicate

Создает массив, заполненный одним значением.

Используется для внутренней реализации [arrayJoin](#).

```
SELECT replicate(x, arr);
```

### Аргументы

- arr — исходный массив. ClickHouse создаёт новый массив такой же длины как исходный и заполняет его значением x.
- x — значение, которым будет заполнен результирующий массив.

### Возвращаемое значение

Массив, заполненный значением x.

Тип: Array.

### Пример

Запрос:

```
SELECT replicate(1, ['a', 'b', 'c']);
```

Ответ:

```
replicate(1, ['a', 'b', 'c'])  
[1,1,1]
```

## filesystemAvailable

Возвращает объём доступного для записи данных места на файловой системе. Он всегда меньше общего свободного места ([filesystemFree](#)), потому что некоторое пространство зарезервировано для нужд операционной системы.

### Синтаксис

```
filesystemAvailable()
```

### Возвращаемое значение

- Объём доступного для записи данных места в байтах.

Тип: [UInt64](#).

### Пример

Запрос:

```
SELECT formatReadableSize(filesystemAvailable()) AS "Available space", toTypeName(filesystemAvailable()) AS  
"Type";
```

Ответ:

```
Available space Type  
30.75 GiB UInt64
```

## filesystemFree

Возвращает объём свободного места на файловой системе. Смотрите также [filesystemAvailable](#).

### Синтаксис

```
filesystemFree()
```

### Возвращаемое значение

- Объем свободного места в байтах.

Тип: [UInt64](#).

### Пример

Запрос:

```
SELECT formatReadableSize(filesystemFree()) AS "Free space", toTypeName(filesystemFree()) AS "Type";
```

Результат:

Free space	Type
32.39 GiB	UInt64

## filesystemCapacity

Возвращает информацию о ёмкости файловой системы в байтах. Для оценки должен быть настроен [путь](#) к каталогу с данными.

### Синтаксис

```
filesystemCapacity()
```

### Возвращаемое значение

- Информация о ёмкости файловой системы в байтах.

Тип: [UInt64](#).

### Пример

Запрос:

```
SELECT formatReadableSize(filesystemCapacity()) AS "Capacity", toTypeName(filesystemCapacity()) AS "Type"
```

Результат:

Capacity	Type
39.32 GiB	UInt64

## initializeAggregation

Вычисляет результат агрегатной функции для каждой строки. Предназначена для инициализации агрегатных функций с комбинатором [-State](#). Может быть полезна для создания состояний агрегатных функций для последующей их вставки в столбцы типа [AggregateFunction](#) или использования в качестве значений по-умолчанию.

### Синтаксис

```
initializeAggregation (aggregate_function, arg1, arg2, ..., argN)
```

### Аргументы

- `aggregate_function` — название агрегатной функции, состояние которой нужно создать. [String](#).
- `arg` — аргументы, которые передаются в агрегатную функцию.

### Возвращаемое значение

- В каждой строке результат агрегатной функции, примененной к аргументам из этой строки.

Тип возвращаемого значения такой же, как и у функции, переданной первым аргументом.

### Пример

Запрос:

```
SELECT uniqMerge(state) FROM (SELECT initializeAggregation('uniqState', number % 3) AS state FROM numbers(10000));
```

Результат:

```
└─uniqMerge(state)─  
   3 |
```

Запрос:

```
SELECT finalizeAggregation(state), toTypeName(state) FROM (SELECT initializeAggregation('sumState', number % 3) AS state FROM numbers(5));
```

Результат:

```
finalizeAggregation(state)─toTypeName(state)─────────  
  0 | AggregateFunction(sum, UInt8) |  
  1 | AggregateFunction(sum, UInt8) |  
  2 | AggregateFunction(sum, UInt8) |  
  0 | AggregateFunction(sum, UInt8) |  
  1 | AggregateFunction(sum, UInt8) |
```

Пример с движком таблиц AggregatingMergeTree и столбцом типа AggregateFunction:

```
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)))
```

## Смотрите также

- [arrayReduce](#)

## finalizeAggregation

Принимает состояние агрегатной функции. Возвращает результат агрегирования (или конечное состояние при использовании комбинатора [-State](#)).

### Синтаксис

```
finalizeAggregation(state)
```

### Аргументы

- state — состояние агрегатной функции. [AggregateFunction](#).

### Возвращаемые значения

- Значения, которые были агрегированы.

Тип: соответствует типу агрегируемых значений.

## Примеры

Запрос:

```
SELECT finalizeAggregation(( SELECT countState(number) FROM numbers(10)));
```

Результат:

```
finalizeAggregation(_subquery16)─  
10 |
```

Запрос:

```
SELECT finalizeAggregation(( SELECT sumState(number) FROM numbers(10)));
```

Результат:

```
finalizeAggregation(_subquery20)─  
45 |
```

Обратите внимание, что значения NULL игнорируются.

Запрос:

```
SELECT finalizeAggregation(arrayReduce('anyState', [NULL, 2, 3]));
```

Результат:

```
finalizeAggregation(arrayReduce('anyState', [NULL, 2, 3]))─  
2 |
```

Комбинированный пример:

Запрос:

```
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);
```

Результат:

number	one	row_sum	cumulative_sum
0	0	0	0
1	1	1	1
2	2	3	3
3	3	6	6
4	4	10	10
5	5	15	15
6	6	21	21
7	7	28	28
8	8	36	36
9	9	45	45

## Смотрите также

- [arrayReduce](#)
- [initializeAggregation](#)

## runningAccumulate

Накапливает состояния агрегатной функции для каждой строки блока данных.

### Warning

Функция обнуляет состояние для каждого нового блока.

## Синтаксис

```
runningAccumulate(agg_state[, grouping])
```

## Аргументы

- `agg_state` — состояние агрегатной функции. [AggregateFunction](#).
- `grouping` — ключ группировки. Опциональный параметр. Состояние функции обнуляется, если значение `grouping` меняется. Параметр может быть любого [поддерживаемого типа данных](#), для которого определен оператор равенства.

## Возвращаемое значение

- Каждая результирующая строка содержит результат агрегатной функции, накопленный для всех входных строк от 0 до текущей позиции. `runningAccumulate` обнуляет состояния для каждого нового блока данных или при изменении значения `grouping`.

Тип зависит от используемой агрегатной функции.

## Примеры

Рассмотрим примеры использования `runningAccumulate` для нахождения кумулятивной суммы чисел без и с группировкой.

Запрос:

```
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);
```

Результат:

k	res
0	0
1	1
2	3
3	6
4	10
5	15
6	21
7	28
8	36
9	45

Подзапрос формирует `sumState` для каждого числа от 0 до 9. `sumState` возвращает состояние функции `sum`, содержащее сумму одного числа.

Весь запрос делает следующее:

- Для первой строки `runningAccumulate` берет `sumState(0)` и возвращает 0.
- Для второй строки функция объединяет `sumState (0)` и `sumState (1)`, что приводит к `sumState (0 + 1)`, и возвращает в результате 1.
- Для третьей строки функция объединяет `sumState (0 + 1)` и `sumState (2)`, что приводит к `sumState (0 + 1 + 2)`, и в результате возвращает 3.
- Действия повторяются до тех пор, пока не закончится блок.

В следующем примере показано использование параметра `grouping`:

Запрос:

```
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
);
```

Результат:

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

Как вы можете видеть, `runningAccumulate` объединяет состояния для каждой группы строк отдельно.

# joinGet

Функция позволяет извлекать данные из таблицы таким же образом как из [словаря](#).

Получает данные из таблиц [Join](#) по ключу.

Поддерживаются только таблицы, созданные с `ENGINE = Join(ANY, LEFT, <join_keys>)`.

## Синтаксис

```
joinGet(join_storage_table_name, `value_column`, join_keys)
```

## Аргументы

- `join_storage_table_name` — [идентификатор](#), который указывает, откуда производится выборка данных. Поиск по идентификатору осуществляется в базе данных по умолчанию (см. конфигурацию `default_database`). Чтобы переопределить базу данных по умолчанию, используйте команду `USE db_name`, или укажите базу данных и таблицу через разделитель `db_name.db_table`, см. пример.
- `value_column` — столбец, из которого нужно произвести выборку данных.
- `join_keys` — список ключей, по которым производится выборка данных.

## Возвращаемое значение

Возвращает значение по списку ключей.

Если значения не существует в исходной таблице, вернется `0` или `null` в соответствии с настройками `join_use_nulls`.

Подробнее о настройке `join_use_nulls` в [операциях Join](#).

## Пример

Входная таблица:

```
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

Запрос:

```
SELECT joinGet(db_test.id_val,'val',toUInt32(number)) from numbers(4) SETTINGS join_use_nulls = 1
```

Результат:

```
joinGet(db_test.id_val, 'val', toUInt32(number))
```

0
11
12
0

## modelEvaluate(model\_name, ...)

Оценивает внешнюю модель.

Принимает на вход имя и аргументы модели. Возвращает Float64.

## throwIf(x[, custom\_message])

Бросает исключение, если аргумент не равен нулю.

custom\_message - необязательный параметр, константная строка, задает текст сообщения об ошибке.

```
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

Возвращает свой аргумент. Используется для отладки и тестирования, позволяет отменить использование индекса, и получить результат и производительность полного сканирования таблицы. Это работает, потому что оптимизатор запросов не может «заглянуть» внутрь функции identity.

### Синтаксис

```
identity(x)
```

### Пример

Query:

```
SELECT identity(42)
```

Результат:

identity(42)
42

## randomPrintableASCII

Генерирует строку со случайным набором печатных символов **ASCII**.

### Синтаксис

```
randomPrintableASCII(length)
```

## Аргументы

- length — длина результирующей строки. Положительное целое число.

Если передать `length < 0`, то поведение функции не определено.

## Возвращаемое значение

- Строка со случайным набором печатных символов [ASCII](#).

Тип: [String](#)

## Пример

```
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"SUF D	30
2	/"+"<"wUTh:=Ljj Vm!c&hl*m#XTfzz	30

## randomString

Генерирует бинарную строку заданной длины, заполненную случайными байтами (в том числе нулевыми).

## Синтаксис

```
randomString(length)
```

## Аргументы

- length — длина строки. Положительное целое число.

## Возвращаемое значение

- Строка, заполненная случайными байтами.

Тип: [String](#).

## Пример

Запрос:

```
SELECT randomString(30) AS str, length(str) AS len FROM numbers(2) FORMAT Vertical;
```

Ответ:

Row 1:  
\_\_\_\_\_  
str: 3 G : pT ?w ti k aV f6  
len: 30

Row 2:  
\_\_\_\_\_  
str: 9 ,] ^ ) ]?? 8  
len: 30

## Смотрите также

- [generateRandom](#)
- [randomPrintableASCII](#)

# randomFixedString

Генерирует бинарную строку заданной длины, заполненную случайными байтами, включая нулевые.

## Синтаксис

```
randomFixedString(length);
```

## Аргументы

- `length` — длина строки в байтах. [UInt64](#).

## Возвращаемое значение

- Стока, заполненная случайными байтами.

Тип: [FixedSize](#).

## Пример

Запрос:

```
SELECT randomFixedString(13) as rnd, toTypeName(rnd)
```

Результат:

rnd	toTypeName(randomFixedString(13))
j█h@HiZ█	FixedString(13)

# randomStringUTF8

Генерирует строку заданной длины со случайными символами в кодировке UTF-8.

## Синтаксис

```
randomStringUTF8(length)
```

## Аргументы

- `length` — длина итоговой строки в кодовых точках. [UInt64](#).

## Возвращаемое значение

- Случайная строка в кодировке UTF-8.

Тип: [String](#).

## Пример

Запрос:

```
SELECT randomStringUTF8(13)
```

## Результат:

## getSetting

Возвращает текущее значение **пользовательской настройки**.

## Синтаксис

```
getSetting('custom_setting')
```

## Параметр

- `custom_setting` — название настройки. `String`.

## Возвращаемое значение

- Текущее значение пользовательской настройки.

## Пример

```
SET custom_a = 123;  
SELECT getSetting('custom_a');
```

## Результат

123

**См. также**

- #### ■ Пользовательские настройки

## isDecimalOverflow

Проверяет, находится ли число **Decimal** вне собственной (или заданной) области значений.

## Синтаксис

`isDecimalOverflow(d, [p])`

## Аргументы

- d — число. [Decimal](#).
  - p — точность. Необязательный параметр. Если опущен, используется исходная точность первого аргумента. Использование этого параметра может быть полезно для извлечения данных в другую СУБД или файл. [UInt8](#).

## Возвращаемое значение

- 1 — число имеет больше цифр, чем позволяет точность.

- 0 — число удовлетворяет заданной точности.

## Пример

Запрос:

```
SELECT isDecimalOverflow(toDecimal32(1000000000, 0), 9),
       isDecimalOverflow(toDecimal32(1000000000, 0)),
       isDecimalOverflow(toDecimal32(-1000000000, 0), 9),
       isDecimalOverflow(toDecimal32(-1000000000, 0));
```

Результат:

```
1 1 1 1
```

## countDigits

Возвращает количество десятичных цифр, необходимых для представления значения.

### Синтаксис

```
countDigits(x)
```

### Аргументы

- x — целое или дробное число.

### Возвращаемое значение

Количество цифр.

Тип: UInt8.

!!! note "Примечание"

Для Decimal значений учитывается их масштаб: вычисляется результат по базовому целочисленному типу, полученному как `(value * scale)`. Например: `countDigits(42) = 2`, `countDigits(42.000) = 5`, `countDigits(0.04200) = 4`. То есть вы можете проверить десятичное переполнение для Decimal64 с помощью `countDecimal(x) > 18`. Это медленный вариант `isDecimalOverflow`.

## Пример

Запрос:

```
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));
```

Результат:

```
10 10 19 19 39 39
```

## errorCodeToName

### Возвращаемое значение

- Название переменной для кода ошибки.

Тип: [LowCardinality\(String\)](#).

## Синтаксис

```
errorCodeToName(1)
```

Результат:

```
UNSUPPORTED_METHOD
```

## tcpPort

Возвращает номер TCP порта, который использует сервер для [нативного протокола](#).

## Синтаксис

```
tcpPort()
```

## Аргументы

- Нет.

## Возвращаемое значение

- Номер TCP порта.

Тип: [UInt16](#).

## Пример

Запрос:

```
SELECT tcpPort();
```

Результат:

```
tcpPort() 9000 |
```

## Смотрите также

- [tcp\\_port](#)

## currentProfiles

Возвращает список [профилей настроек](#) для текущего пользователя.

Для изменения текущего профиля настроек может быть использована команда `SET PROFILE`. Если команда `SET PROFILE` не применялась, функция возвращает профили, указанные при определении текущего пользователя (см. [CREATE USER](#)).

## Синтаксис

```
currentProfiles()
```

## **Возвращаемое значение**

- Список профилей настроек для текущего пользователя.

Тип: [Array\(String\)](#).

## **enabledProfiles**

Возвращает профили настроек, назначенные пользователю как явно, так и неявно. Явно назначенные профили — это те же профили, которые возвращает функция [currentProfiles](#). Неявно назначенные профили включают родительские профили других назначенных профилей; профили, назначенные с помощью предоставленных ролей; профили, назначенные с помощью собственных настроек; основной профиль по умолчанию (см. секцию `default_profile` в основном конфигурационном файле сервера).

### **Синтаксис**

```
enabledProfiles()
```

## **Возвращаемое значение**

- Список доступных профилей для текущего пользователя.

Тип: [Array\(String\)](#).

## **defaultProfiles**

Возвращает все профили, указанные при объявлении текущего пользователя (см. [CREATE USER](#))

### **Синтаксис**

```
defaultProfiles()
```

## **Возвращаемое значение**

- Список профилей по умолчанию.

Тип: [Array\(String\)](#).

## **currentRoles**

Возвращает список текущих ролей для текущего пользователя. Список ролей пользователя можно изменить с помощью выражения [SET ROLE](#). Если выражение SET ROLE не использовалось, данная функция возвращает тот же результат, что и функция [defaultRoles](#).

### **Синтаксис**

```
currentRoles()
```

## **Возвращаемое значение**

- Список текущих ролей для текущего пользователя.

Тип: [Array\(String\)](#).

## **enabledRoles**

Возвращает имена текущих ролей, а также ролей, которые разрешено использовать текущему пользователю путем назначения привилегий.

## Синтаксис

```
enabledRoles()
```

### Возвращаемое значение

- Список доступных ролей для текущего пользователя.

Тип: [Array\(String\)](#).

## defaultRoles

Возвращает имена ролей, которые задаются по умолчанию для текущего пользователя при входе в систему. Изначально это все роли, которые разрешено использовать текущему пользователю (см. [GRANT](#)). Список ролей по умолчанию может быть изменен с помощью выражения [SET DEFAULT ROLE](#).

## Синтаксис

```
defaultRoles()
```

### Возвращаемое значение

- Список ролей по умолчанию.

Тип: [Array\(String\)](#).

## getServerPort

Возвращает номер порта сервера. Если порт не используется сервером, генерируется исключение.

## Синтаксис

```
getServerPort(port_name)
```

## Аргументы

- `port_name` — имя порта сервера. [String](#). Возможные значения:

- 'tcp\_port'
- 'tcp\_port\_secure'
- 'http\_port'
- 'https\_port'
- 'interserver\_http\_port'
- 'interserver\_https\_port'
- 'mysql\_port'
- 'postgresql\_port'
- 'grpc\_port'
- 'prometheus.port'

### **Возвращаемое значение**

- Номер порта сервера.

Тип: [UInt16](#).

### **Пример**

Запрос:

```
SELECT getServerPort('tcp_port');
```

Результат:

```
getServerPort('tcp_port')—  
9000
```

## **queryID**

Возвращает идентификатор текущего запроса. Другие параметры запроса могут быть извлечены из системной таблицы [system.query\\_log](#) через `query_id`.

В отличие от [initialQueryID](#), функция `queryID` может возвращать различные значения для разных шардов (см. пример).

### **Синтаксис**

```
queryID()
```

### **Возвращаемое значение**

- Идентификатор текущего запроса.

Тип: [String](#)

### **Пример**

Запрос:

```
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());
```

Результат:

count()
3

## initialQueryID

Возвращает идентификатор родительского запроса. Другие параметры запроса могут быть извлечены из системной таблицы [system.query\\_log](#) через `initial_query_id`.

В отличие от `queryID`, функция `initialQueryID` возвращает одинаковые значения для разных шардов (см. пример).

### Синтаксис

```
initialQueryID()
```

### Возвращаемое значение

- Идентификатор родительского запроса.

Тип: [String](#)

### Пример

Запрос:

```
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());
```

Результат:

count()
1

## shardNum

Возвращает индекс шарда, который обрабатывает часть данных распределенного запроса.  
Индексы начинаются с 1.

Если запрос не распределенный, то возвращается значение 0.

### Синтаксис

```
shardNum()
```

### Возвращаемое значение

- индекс шарда или константа 0.

Тип: [UInt32](#).

## Пример

В примере ниже используется конфигурация с двумя шардами. На каждом шарде выполняется запрос к таблице [system.one](#).

Запрос:

```
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;
```

Результат:

dummy	shardNum()	shardCount()
0	2	2
0	1	2

## См. также

- Табличный движок [Distributed](#)

## shardCount

Возвращает общее количество шардов для распределенного запроса.

Если запрос не распределенный, то возвращается значение 0.

## Синтаксис

```
shardCount()
```

## Возвращаемое значение

- Общее количество шардов или 0.

Тип: [UInt32](#).

## См. также

- Пример использования функции [shardNum\(\)](#) также содержит вызов [shardCount\(\)](#).

## getOSKernelVersion

Возвращает строку с текущей версией ядра ОС.

## Синтаксис

```
getOSKernelVersion()
```

## Аргументы

- Нет.

## Возвращаемое значение

- Текущая версия ядра ОС.

Тип: **String**.

## Пример

Запрос:

```
SELECT getOSKernelVersion();
```

Результат:

```
getOSKernelVersion()——  
Linux 4.15.0-55-generic |
```

# [экспериментально] Функции для работы с естественным языком

## Предупреждение

Сейчас использование функций для работы с естественным языком является экспериментальной возможностью. Чтобы использовать данные функции, включите настройку `allow_experimental_nlp_functions = 1`.

## stem

Данная функция проводит стемминг заданного слова.

### Синтаксис

```
stem('language', word)
```

### Аргументы

- `language` — Язык, правила которого будут применены для стемминга. Допускается только нижний регистр. **String**.
- `word` — Слово подлежащее стеммингу. Допускается только нижний регистр. **String**.

### Examples

Query:

```
SELECT 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

Данная функция проводит лемматизацию для заданного слова. Для работы лемматизатора необходимы словари, которые можно найти [здесь](#).

## Синтаксис

```
lemmatize('language', word)
```

## Аргументы

- `language` — Язык, правила которого будут применены для лемматизации. [String](#).
- `word` — Слово, подлежащее лемматизации. Допускается только нижний регистр. [String](#).

## Примеры

Запрос:

```
SELECT lemmatize('en', 'wolves');
```

Результат:

```
└── lemmatize("wolves") └──  
    "wolf" |
```

Конфигурация:

```
<lemmatizers>  
  <lemmatizer>  
    <lang>en</lang>  
    <path>en.bin</path>  
  </lemmatizer>  
</lemmatizers>
```

# synonyms

Находит синонимы к заданному слову. Представлены два типа расширений словарей: `plain` и `wordnet`.

Для работы расширения типа `plain` необходимо указать путь до простого текстового файла, где каждая строка соответствует одному набору синонимов. Слова в данной строке должны быть разделены с помощью пробела или знака табуляции.

Для работы расширения типа `plain` необходимо указать путь до WordNet тезауруса. Тезаурус должен содержать WordNet sense index.

## Синтаксис

```
synonyms('extension_name', word)
```

## Аргументы

- `extension_name` — Название расширения, в котором будет проводиться поиск. [String](#).
- `word` — Слово, которое будет искааться в расширении. [String](#).

## Примеры

Запрос:

```
SELECT synonyms('list', 'important');
```

Результат:

```
synonyms('list', 'important')-----  
['important','big','critical','crucial'] |
```

Конфигурация:

```
<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>
```

## ФУНКЦИИ ШИФРОВАНИЯ

Данные функции реализуют шифрование и расшифровку данных с помощью AES (Advanced Encryption Standard) алгоритма.

Длина ключа зависит от режима шифрования. Он может быть длинной в 16, 24 и 32 байта для режимов шифрования `-128-`, `-196-` и `-256-` соответственно.

Длина инициализирующего вектора всегда 16 байт (лишние байты игнорируются).

Обратите внимание, что до версии Clickhouse 21.1 эти функции работали медленно.

### encrypt

Функция поддерживает шифрование данных следующими режимами:

- 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

### Синтаксис

```
encrypt('mode', 'plaintext', 'key' [, iv, aad])
```

## Аргументы

- `mode` — режим шифрования. `String`.
- `plaintext` — текст, который будет зашифрован. `String`.
- `key` — ключ шифрования. `String`.
- `iv` — инициализирующий вектор. Обязателен для `-gcm` режимов, для остальных режимов необязателен. `String`.
- `aad` — дополнительные аутентифицированные данные. Не шифруются, но влияют на расшифровку. Параметр работает только с `-gcm` режимами. Для остальных вызовет исключение. `String`.

## Возвращаемое значение

- Бинарная зашифрованная строка. `String`.

## Примеры

Создадим такую таблицу:

Запрос:

```
CREATE TABLE encryption_test
(
    `comment` String,
    `secret` String
)
ENGINE = Memory;
```

Вставим некоторые данные (замечание: не храните ключи или инициализирующие векторы в базе данных, так как это компрометирует всю концепцию шифрования), также хранение "подсказок" небезопасно и используется только для наглядности:

Запрос:

```
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\n('aes-256-cfb128 with IV', encrypt('aes-256-cfb128', 'Secret', '12345678910121314151617181920212',\n'iviviviviviviv')),\\n\n('aes-256-cbc no IV', encrypt('aes-256-cbc', 'Secret', '12345678910121314151617181920212'));
```

Запрос:

```
SELECT comment, hex(secret) FROM encryption_test;
```

Результат:

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

Пример в режиме `-gcm`:

Запрос:

```
INSERT INTO encryption_test VALUES('aes-256-gcm', encrypt('aes-256-gcm', 'Secret', '12345678910121314151617181920212', 'iviviviviviviv')), \
('aes-256-gcm with AAD', encrypt('aes-256-gcm', 'Secret', '12345678910121314151617181920212', 'iviviviviviviv', 'aad'));

SELECT comment, hex(secret) FROM encryption_test WHERE comment LIKE '%gcm%';
```

Результат:

comment	hex(secret)
aes-256-gcm	A8A3CCBC6426CFEEB60E4EAE03D3E94204C1B09E0254
aes-256-gcm with AAD	A8A3CCBC6426D9A1017A0A932322F1852260A4AD6837

## aes\_encrypt\_mysql

Совместима с шифрованием mysql, результат может быть расшифрован функцией [AES\\_DECRYPT](#).

При одинаковых входящих значениях зашифрованный текст будет совпадать с результатом, возвращаемым функцией `encrypt`. Однако если `key` или `iv` длиннее, чем должны быть, `aes_encrypt_mysql` будет работать аналогично функции `aes_encrypt` в MySQL: свернет ключ и проигнорирует лишнюю часть `iv`.

Функция поддерживает шифрование данных следующими режимами:

- 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

### Синтаксис

```
aes_encrypt_mysql('mode', 'plaintext', 'key' [, iv])
```

### Аргументы

- `mode` — режим шифрования. [String](#).
- `plaintext` — текст, который будет зашифрован. [String](#).
- `key` — ключ шифрования. Если ключ длиннее, чем требует режим шифрования, производится специфичная для MySQL свертка ключа. [String](#).
- `iv` — инициализирующий вектор. Необязателен, учитываются только первые 16 байтов. [String](#).

### Возвращаемое значение

- Бинарная зашифрованная строка. [String](#).

### Примеры

При одинаковых входящих значениях результаты шифрования у функций `encrypt` и `aes_encrypt_mysql` совпадают.

Запрос:

```
SELECT encrypt('aes-256-cfb128', 'Secret', '12345678910121314151617181920212', 'iviviviviviviv') =  
aes_encrypt_mysql('aes-256-cfb128', 'Secret', '12345678910121314151617181920212', 'iviviviviviviv') AS  
ciphertexts_equal;
```

Результат:

```
ciphertexts_equal  
1
```

Функция `encrypt` генерирует исключение, если key или iv длиннее чем нужно:

Запрос:

```
SELECT encrypt('aes-256-cfb128', 'Secret', '123456789101213141516171819202122', 'iviviviviviviv123');
```

Результат:

```
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').
```

Однако функция `aes_encrypt_mysql` в аналогичном случае возвращает результат, который может быть обработан MySQL:

Запрос:

```
SELECT hex(aes_encrypt_mysql('aes-256-cfb128', 'Secret', '123456789101213141516171819202122',  
'iviviviviviviv123')) AS ciphertext;
```

Результат:

```
ciphertext  
24E9E4966469
```

Если передать iv еще длиннее, результат останется таким же:

Запрос:

```
SELECT hex(aes_encrypt_mysql('aes-256-cfb128', 'Secret', '123456789101213141516171819202122',  
'iviviviviviviv123456')) AS ciphertext
```

Результат:

```
ciphertext  
24E9E4966469
```

Это совпадает с результатом, возвращаемым MySQL при таких же входящих значениях:

```
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

Функция расшифровывает зашифрованный текст и может работать в следующих режимах:

- 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

### Синтаксис

```
decrypt('mode', 'ciphertext', 'key' [, iv, aad])
```

### Аргументы

- `mode` — режим шифрования. [String](#).
- `ciphertext` — зашифрованный текст, который будет расшифрован. [String](#).
- `key` — ключ шифрования. [String](#).
- `iv` — инициализирующий вектор. Обязателен для `-gcm` режимов, для остальных режимов опциональный. [String](#).
- `aad` — дополнительные аутентифицированные данные. Текст не будет расшифрован, если это значение неверно. Работает только с `-gcm` режимами. Для остальных вызовет исключение. [String](#).

### Возвращаемое значение

- Расшифрованная строка. [String](#).

### Примеры

Рассмотрим таблицу из примера для функции `encrypt`.

Запрос:

```
SELECT comment, hex(secret) FROM encryption_test;
```

Результат:

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

Теперь попытаемся расшифровать эти данные:

## Запрос:

```
SELECT comment, decrypt('aes-256-cfb128', secret, '12345678910121314151617181920212') as plaintext FROM encryption_test;
```

## Результат:

```

graph TD
    comment[comment] --> aes1[aes-256-cfb128 no IV]
    comment --> aes2[aes-256-cfb128 no IV, different key]
    aes1 --> Secret1[Secret]
    aes1 --> plaintext1[plaintext]
    aes2 --> Secret2[Secret]
    aes2 --> plaintext2[plaintext]
    Secret1 --> aes3[aes-256-cfb128 with IV]
    Secret1 --> aes4[aes-256-cbc no IV]
    Secret2 --> aes5["2?6?~"]
    Secret2 --> aes6["2*4~h3c4w@"]

```

Обратите внимание, что только часть данных была расшифрована верно. Оставшаяся часть расшифрована некорректно, так как при шифровании использовались другие значения `mode`, `key`, или `iv`.

## aes decrypt mysql

Совместима с шифрованием MySQL и может расшифровать данные, зашифрованные функцией [AES\\_ENCRYPT](#).

При одинаковых входящих значениях расшифрованный текст будет совпадать с результатом, возвращаемым функцией `decrypt`. Однако если `key` или `iv` длиннее, чем должны быть, `aes_decrypt_mysql` будет работать аналогично функции `aes_decrypt` в MySQL: свернет ключ и проигнорирует лишнюю часть `iv`.

Функция поддерживает расшифровку данных в следующих режимах:

- 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

## Синтаксис

```
aes_decrypt_mysql('mode' 'ciphertext' 'key' [ 'iv'])
```

## Аргументы

- `mode` — режим шифрования. **String**.
- `ciphertext` — зашифрованный текст, который будет расшифрован. **String**.
- `key` — ключ шифрования. **String**.
- `iv` — инициализирующий вектор. Необязателен. **String**.

## Возвращаемое значение

- Расшифрованная строка. **String**.

## Примеры

Расшифруем данные, которые до этого были зашифрованы в 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)
```

Запрос:

```
SELECT aes_decrypt_mysql('aes-256-cfb128', unhex('24E9E4966469'), '123456789101213141516171819202122',
'iviviviviviviv123456') AS plaintext;
```

Результат:

```
plaintext
Secret
```

## ФУНКЦИИ ДЛЯ РАБОТЫ С КОРТЕЖАМИ `tuple`

Функция, позволяющая сгруппировать несколько столбцов.

Для столбцов, имеющих типы T1, T2, ... возвращает кортеж типа Tuple(T1, T2, ...), содержащий эти столбцы. Выполнение функции ничего не стоит.

Кортежи обычно используются как промежуточное значение в качестве аргумента операторов IN, или для создания списка формальных параметров лямбда-функций. Кортежи не могут быть записаны в таблицу.

С помощью функции реализуется оператор (x, y, ...).

## Синтаксис

```
tuple(x, y, ...)
```

## tupleElement

Функция, позволяющая достать столбец из кортежа.

N - индекс столбца начиная с 1. N должно быть константой. N должно быть целым строго положительным числом не большим размера кортежа.

Выполнение функции ничего не стоит.

С помощью функции реализуется оператор x.N.

### Синтаксис

```
tupleElement(tuple, n)
```

## untuple

Выполняет синтаксическую подстановку элементов [кортежа](#) в место вызова.

### Синтаксис

```
untuple(x)
```

Чтобы пропустить некоторые столбцы в результате запроса, вы можете использовать выражение EXCEPT.

### Аргументы

- x — функция tuple, столбец или кортеж элементов. [Tuple](#).

### Возвращаемое значение

- Нет.

### Примеры

Входная таблица:

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')

Пример использования столбца типа Tuple в качестве параметра функции untuple:

Запрос:

```
SELECT untuple(v6) FROM kv;
```

Результат:

ut_1		ut_2	
33	ab		
44	cd		
55	ef		
66	gh		
77	kl		

Пример использования выражения EXCEPT:

Запрос:

```
SELECT untuple((* EXCEPT (v2, v3),)) FROM kv;
```

Результат:

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')

## Смотрите также

- [Tuple](#)

## tupleHammingDistance

Возвращает [расстояние Хэмминга](#) между двумя кортежами одинакового размера.

### Синтаксис

```
tupleHammingDistance(tuple1, tuple2)
```

### Аргументы

- `tuple1` — первый кортеж. [Tuple](#).
- `tuple2` — второй кортеж. [Tuple](#).

Кортежи должны иметь одинаковый размер и тип элементов.

### Возвращаемое значение

- Расстояние Хэмминга.

Тип: [UInt8](#).

### Примеры

Запрос:

```
SELECT tupleHammingDistance((1, 2, 3), (3, 2, 1)) AS HammingDistance;
```

Результат:

```
└─HammingDistance─  
  2 |
```

Может быть использовано с функциями [MinHash](#) для проверки строк на совпадение:

```
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);
```

Результат:

```
└─HammingDistance─  
  2 |
```

## tupleToNameValuePairs

Приводит именованный кортеж к списку пар (имя, значение). Для Tuple(a T, b T, ..., c T) возвращает Array(Tuple(String, T), ...), где Strings — это названия именованных полей, а T — это соответствующие значения. Все значения в кортеже должны быть одинакового типа.

### Синтаксис

```
tupleToNameValuePairs(tuple)
```

### Аргументы

- tuple — именованный кортеж. [Tuple](#) с любым типом значений.

### Возвращаемое значение

- Список пар (имя, значение).

Тип: [Array\(Tuple\(String, ...\)\)](#).

### Пример

Запрос:

```
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;
```

Результат:

```
└─tupleToNameValuePairs(col)──────────  
  [('user_ID',100),('session_ID',2502)] |  
  [('user_ID',1),('session_ID',100)] |
```

С помощью этой функции можно выводить столбцы в виде строк:

```
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;
```

Результат:

```
arrayJoin(tupleToNameValuePairs(col))—
('CPU',3.3)      |
('Memory',5.5)   |
('Disk',6.6)     |
```

Если в функцию передается обычный кортеж, ClickHouse использует индексы значений в качестве имен:

```
SELECT tupleToNameValuePairs(tuple(3, 2, 1));
```

Результат:

```
tupleToNameValuePairs(tuple(3, 2, 1))—
[('1',3),('2',2),('3',1)] |
```

```
### tuplePlus {#tupleplus}
```

Вычисляет сумму соответствующих значений двух кортежей одинакового размера.

**\*\*Синтаксис\*\***

```
```sql
tuplePlus(tuple1, tuple2)
```

Синоним: `vectorSum`.

## Аргументы

- `tuple1` — первый кортеж. [Tuple](#).
- `tuple2` — второй кортеж. [Tuple](#).

## Возвращаемое значение

- Кортеж с суммами.

Тип: [Tuple](#).

## Пример

Запрос:

```
SELECT tuplePlus((1, 2), (2, 3));
```

Результат:

```
tuplePlus((1, 2), (2, 3))—
(3,5) |
```

# tupleMinus

Вычисляет разность соответствующих значений двух кортежей одинакового размера.

## Синтаксис

```
tupleMinus(tuple1, tuple2)
```

Синоним: vectorDifference.

## Аргументы

- tuple1 — первый кортеж. [Tuple](#).
- tuple2 — второй кортеж. [Tuple](#).

## Возвращаемое значение

- Кортеж с разностями.

Тип: [Tuple](#).

## Пример

Запрос:

```
SELECT tupleMinus((1, 2), (2, 3));
```

Результат:

```
tupleMinus((1, 2), (2, 3))—  
(-1,-1) |
```

# tupleMultiply

Вычисляет произведение соответствующих значений двух кортежей одинакового размера.

## Синтаксис

```
tupleMultiply(tuple1, tuple2)
```

## Аргументы

- tuple1 — первый кортеж. [Tuple](#).
- tuple2 — второй кортеж. [Tuple](#).

## Возвращаемое значение

- Кортеж с произведениями.

Тип: [Tuple](#).

## Пример

Запрос:

```
SELECT tupleMultiply((1, 2), (2, 3));
```

Результат:

```
tupleMultiply((1, 2), (2, 3)) └  
| (2,6) └
```

## tupleDivide

Вычисляет частное соответствующих значений двух кортежей одинакового размера. Обратите внимание, что при делении на ноль возвращается значение `inf`.

### Синтаксис

```
tupleDivide(tuple1, tuple2)
```

### Аргументы

- `tuple1` — первый кортеж. [Tuple](#).
- `tuple2` — второй кортеж. [Tuple](#).

### Возвращаемое значение

- Кортеж с частными.

Тип: [Tuple](#).

### Пример

Запрос:

```
SELECT tupleDivide((1, 2), (2, 3));
```

Результат:

```
tupleDivide((1, 2), (2, 3)) └  
| (0.5,0.6666666666666666) └
```

## tupleNegate

Применяет отрицание ко всем значениям кортежа.

### Синтаксис

```
tupleNegate(tuple)
```

### Аргументы

- `tuple` — кортеж. [Tuple](#).

### Возвращаемое значение

- Кортеж с результатом отрицания.

Тип: [Tuple](#).

## Пример

Запрос:

```
SELECT tupleNegate((1, 2));
```

Результат:

```
tupleNegate((1, 2))  
(-1,-2)
```

## tupleMultiplyByNumber

Возвращает кортеж, в котором значения всех элементов умножены на заданное число.

### Синтаксис

```
tupleMultiplyByNumber(tuple, number)
```

### Аргументы

- `tuple` — кортеж. [Tuple](#).
- `number` — множитель. [Int/UInt](#), [Float](#) или [Decimal](#).

### Возвращаемое значение

- Кортеж с результатами умножения на число.

Тип: [Tuple](#).

## Пример

Запрос:

```
SELECT tupleMultiplyByNumber((1, 2), -2.1);
```

Результат:

```
tupleMultiplyByNumber((1, 2), -2.1)  
(-2.1,-4.2)
```

## tupleDivideByNumber

Возвращает кортеж, в котором значения всех элементов поделены на заданное число. Обратите внимание, что при делении на ноль возвращается значение `inf`.

### Синтаксис

```
tupleDivideByNumber(tuple, number)
```

### Аргументы

- `tuple` — кортеж. [Tuple](#).
- `number` — делитель. [Int/UInt](#), [Float](#) or [Decimal](#).

### Возвращаемое значение

- Кортеж с результатами деления на число.

Тип: [Tuple](#).

### Пример

Запрос:

```
SELECT tupleDivideByNumber((1, 2), 0.5);
```

Результат:

```
tupleDivideByNumber((1, 2), 0.5)─  
(2,4) ─
```

## dotProduct

Вычисляет скалярное произведение двух кортежей одинакового размера.

### Синтаксис

```
dotProduct(tuple1, tuple2)
```

Синоним: `scalarProduct`.

### Аргументы

- `tuple1` — первый кортеж. [Tuple](#).
- `tuple2` — второй кортеж. [Tuple](#).

### Возвращаемое значение

- Скалярное произведение.

Тип: [Int/UInt](#), [Float](#) или [Decimal](#).

### Пример

Запрос:

```
SELECT dotProduct((1, 2), (2, 3));
```

Результат:

```
dotProduct((1, 2), (2, 3))─  
8 ─
```

## L1Norm

Вычисляет сумму абсолютных значений кортежа.

## Синтаксис

```
L1Norm(tuple)
```

Синоним: normL1.

## Аргументы

- tuple — кортеж. [Tuple](#).

## Возвращаемое значение

- L1-норма или [расстояние городских кварталов](#).

Тип: [UInt](#), [Float](#) или [Decimal](#).

## Пример

Запрос:

```
SELECT L1Norm((1, 2));
```

Результат:

```
+-----+  
| L1Norm((1, 2)) |  
+-----+  
| 3 |  
+-----+
```

## L2Norm

Вычисляет квадратный корень из суммы квадратов значений кортежа.

## Синтаксис

```
L2Norm(tuple)
```

Синоним: normL2.

## Аргументы

- tuple — кортеж. [Tuple](#).

## Возвращаемое значение

- L2-норма или [Евклидово расстояние](#).

Тип: [Float](#).

## Пример

Запрос:

```
SELECT L2Norm((1, 2));
```

Результат:

```
L2Norm((1, 2))
```

```
2.23606797749979 |
```

## LinfNorm

Вычисляет максимум из абсолютных значений кортежа.

### Синтаксис

```
LinfNorm(tuple)
```

Синоним: `normLinf`.

### Аргументы

- `tuple` — кортеж. [Tuple](#).

### Возвращаемое значение

- Linf-норма или максимальное абсолютное значение.

Тип: [Float](#).

### Пример

Запрос:

```
SELECT LinfNorm((1, -2));
```

Результат:

```
LinfNorm((1, -2))
```

```
2 |
```

## LpNorm

Возвращает корень степени  $p$  из суммы абсолютных значений кортежа, возведенных в степень  $p$ .

### Синтаксис

```
LpNorm(tuple, p)
```

Синоним: `normLp`.

### Аргументы

- `tuple` — кортеж. [Tuple](#).
- `p` — степень. Возможные значение: любое число из промежутка  $[1; \infty)$ . [UInt](#) или [Float](#).

### Возвращаемое значение

- L $p$ -норма

Тип: [Float](#).

## Пример

Запрос:

```
SELECT LpNorm((1, -2), 2);
```

Результат:

```
LpNorm((1, -2), 2) └  
2.23606797749979 |
```

## L1Distance

Вычисляет расстояние между двумя точками (значения кортежей — координаты точек) в пространстве L1 ([расстояние городских кварталов](#)).

### Синтаксис

```
L1Distance(tuple1, tuple2)
```

Синоним: `distanceL1`.

### Аргументы

- `tuple1` — первый кортеж. [Tuple](#).
- `tuple2` — второй кортеж. [Tuple](#).

### Возвращаемое значение

- Расстояние в норме L1.

Тип: [Float](#).

## Пример

Запрос:

```
SELECT L1Distance((1, 2), (2, 3));
```

Результат:

```
L1Distance((1, 2), (2, 3)) └  
2 |
```

## L2Distance

Вычисляет расстояние между двумя точками (значения кортежей — координаты точек) в пространстве L2 ([Евклидово расстояние](#)).

### Синтаксис

```
L2Distance(tuple1, tuple2)
```

Синоним: distanceL2.

## Аргументы

- tuple1 — первый кортеж. **Tuple**.
- tuple2 — второй кортеж. **Tuple**.

## Возвращаемое значение

- Расстояние в норме L2.

Тип: **Float**.

## Пример

Запрос:

```
SELECT L2Distance((1, 2), (2, 3));
```

Результат:

```
L2Distance((1, 2), (2, 3))—  
1.4142135623730951 |
```

## LinfDistance

Вычисляет расстояние между двумя точками (значения кортежей — координаты точек) в пространстве L\_{inf}.

## Синтаксис

```
LinfDistance(tuple1, tuple2)
```

Синоним: distanceLinf.

## Аргументы

- tuple1 — первый кортеж. **Tuple**.
- tuple2 — второй кортеж. **Tuple**.

## Возвращаемые значения

- Расстояние в норме Linf.

Тип: **Float**.

## Пример

Запрос:

```
SELECT LinfDistance((1, 2), (2, 3));
```

Результат:

```
LinfDistance((1, 2), (2, 3))  
 1 |
```

## LpDistance

Вычисляет расстояние между двумя точками (значения кортежей — координаты точек) в пространстве L<sub>p</sub>.

### Синтаксис

```
LpDistance(tuple1, tuple2, p)
```

Синоним: `distanceLp`.

### Аргументы

- `tuple1` — первый кортеж. [Tuple](#).
- `tuple2` — второй кортеж. [Tuple](#).
- `p` — степень. Возможные значение: любое число из промежутка [1;inf). [UInt](#) или [Float](#).

### Возвращаемое значение

- Расстояние в норме L<sub>p</sub>.

Тип: [Float](#).

### Пример

Запрос:

```
SELECT LpDistance((1, 2), (2, 3), 3);
```

Результат:

```
LpDistance((1, 2), (2, 3), 3)  
 1.2599210498948732 |
```

## L1Normalize

Вычисляет единичный вектор для исходного вектора (значения кортежа — координаты вектора) в пространстве L<sub>1</sub> ([расстояние городских кварталов](#)).

### Синтаксис

```
L1Normalize(tuple)
```

Синоним: `normalizeL1`.

### Аргументы

- `tuple` — [Tuple](#).

### Возвращаемое значение

- Единичный вектор.

Тип: кортеж **Tuple** значений **Float**.

### Пример

Запрос:

```
SELECT L1Normalize((1, 2));
```

Результат:

```
L1Normalize((1, 2))—  
(0.33333333333333, 0.66666666666666) |
```

## L2Normalize

Вычисляет единичный вектор для исходного вектора (значения кортежа — координаты вектора) в пространстве L2 ([Евклидово пространство](#)).

### Синтаксис

```
L2Normalize(tuple)
```

Синоним: `normalizeL1`.

### Аргументы

- `tuple` — кортеж **Tuple**.

### Возвращаемое значение

- Единичный вектор.

Тип: кортеж **Tuple** значений **Float**.

### Пример

Запрос:

```
SELECT L2Normalize((3, 4));
```

Результат:

```
L2Normalize((3, 4))—  
(0.6, 0.8) |
```

## LinfNormalize

Вычисляет единичный вектор для исходного вектора (значения кортежа — координаты вектора) в пространстве L\_{inf}.

### Синтаксис

```
LinfNormalize(tuple)
```

Синоним: `normalizeLinf`.

## Аргументы

- `tuple` — кортеж. [Tuple](#).

## Возвращаемое значение

- Единичный вектор.

Тип: кортеж [Tuple](#) значений [Float](#).

## Пример

Запрос:

```
SELECT LinfNormalize((3, 4));
```

Результат:

```
└─LinfNormalize((3, 4))─  
   |  
(0.75,1) ─
```

# LpNormalize

Вычисляет единичный вектор для исходного вектора (значения кортежа — координаты вектора) в пространстве  $L_p$ .

## Синтаксис

```
LpNormalize(tuple, p)
```

Синоним: `normalizeLp`.

## Аргументы

- `tuple` — кортеж. [Tuple](#).
- `p` — степень. Возможные значение: любое число из промежутка  $[1; \infty)$ . [UInt](#) или [Float](#).

## Возвращаемое значение

- Единичный вектор.

Тип: кортеж [Tuple](#) значений [Float](#).

## Пример

Запрос:

```
SELECT LpNormalize((3, 4),5);
```

Результат:

```
LpNormalize((3, 4), 5)  
[0.7187302630182624, 0.9583070173576831] |
```

## cosineDistance

Вычисляет косинусную разницу двух векторов (значения кортежей — координаты векторов). Чем меньше возвращаемое значение, тем больше сходство между векторами.

## **Синтаксис**

`cosineDistance(tuple1, tuple2)`

## Аргументы

- `tuple1` — первый кортеж. **Tuple**.
  - `tuple2` — второй кортеж. **Tuple**.

## Возвращаемые значения

- Разность между единицей и косинусом угла между векторами.

Тип: **Float**.

## Пример

## Запрос:

```
SELECT cosineDistance((1, 2), (2, 3));
```

## Результат:

```
cosineDistance((1, 2), (2, 3))  
0.007722123286332261
```

## Агрегатные функции

Агрегатные функции работают в [привычном](#) для специалистов по базам данных смысле.

ClickHouse поддерживает также:

- **Параметрические агрегатные функции**, которые помимо столбцов принимают и другие параметры.
  - **Комбинаторы**, которые изменяют поведение агрегатных функций.

## Обработка NULL

При агрегации все **NULL** пропускаются.

## Примеры

Рассмотрим таблицу:

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

Выполним суммирование значений в столбце y:

```
SELECT sum(y) FROM t_null_big
```

```
sum(y)  
7 |
```

Теперь с помощью функции `groupArray` сформируем массив из столбца y:

```
SELECT groupArray(y) FROM t_null_big
```

```
groupArray(y)  
[2,2,3] |
```

`groupArray` не включает `NULL` в результирующий массив.

## count

Вычисляет количество строк или не `NULL` значений.

ClickHouse поддерживает следующие виды синтаксиса для `count`:

- `count(expr)` или `COUNT(DISTINCT expr)`.
- `count()` или `COUNT(*)`. Синтаксис `count()` специфичен для ClickHouse.

### Аргументы

Функция может принимать:

- Ноль параметров.
- Одно **выражение**.

### Возвращаемое значение

- Если функция вызывается без параметров, она вычисляет количество строк.
- Если передаётся **выражение**, то функция подсчитывает количество раз, когда выражение не равно `NULL`. Если выражение имеет тип `Nullable`, то результат `count` не становится `Nullable`. Функция возвращает 0, если выражение равно `NULL` для всех строк.

В обоих случаях тип возвращаемого значения `UInt64`.

### Подробности

ClickHouse поддерживает синтаксис `COUNT(DISTINCT ...)`. Поведение этой конструкции зависит от настройки `count_distinctImplementation`. Она определяет, какая из функций `uniq*` используется для выполнения операции. По умолчанию — функция `uniqExact`.

Запрос `SELECT count() FROM table` оптимизирован по умолчанию с использованием метаданных из MergeTree. Если вы хотите управлять безопасностью на уровне строк, отключите оптимизацию при помощи настройки `optimize_trivial_count_query`.

При этом запрос `SELECT count(nullable_column) FROM table` может быть оптимизирован включением настройки `optimize_functions_to_subcolumns`. При `optimize_functions_to_subcolumns = 1` функция читает только подстолбец `null` вместо чтения всех данных столбца. Запрос `SELECT count(n) FROM table` преобразуется к запросу `SELECT sum(NOT n.null) FROM table`.

## Примеры

Пример 1:

```
SELECT count() FROM t
```

```
count()
5 |
```

Пример 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 |
```

Этот пример показывает, что `count(DISTINCT num)` выполняется с помощью функции `uniqExact` в соответствии со значением настройки `count_distinctImplementation`.

## 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

Вычисляет сумму.

Работает только для чисел.

## avg

Вычисляет среднее арифметическое.

### Синтаксис

```
avg(x)
```

### Аргументы

- $x$  — входное значение типа Integer, Float или Decimal.

### Возвращаемое значение

- среднее арифметическое, всегда типа Float64.
- NaN, если входное значение  $x$  — пустое.

### Пример

Запрос:

```
SELECT avg(x) FROM values('x Int8', 0, 1, 2, 3, 4, 5);
```

Результат:

```
avg(x)
2.5 |
```

## Пример

Создайте временную таблицу:

Запрос:

```
CREATE table test (t UInt8) ENGINE = Memory;
```

Выполните запрос:

```
SELECT avg(t) FROM test;
```

Результат:

```
avg(x)
nan
```

## any

Выбирает первое попавшееся значение.

Порядок выполнения запроса может быть произвольным и даже каждый раз разным, поэтому результат данной функции недетерминирован.

Для получения детерминированного результата, можно использовать функции `min` или `max` вместо `any`.

В некоторых случаях, вы всё-таки можете рассчитывать на порядок выполнения запроса. Это - случаи, когда `SELECT` идёт из подзапроса, в котором используется `ORDER BY`.

При наличии в запросе `SELECT` секции `GROUP BY` или хотя бы одной агрегатной функции, ClickHouse (в отличие от, например, MySQL) требует, чтобы все выражения в секциях `SELECT`, `HAVING`, `ORDER BY` вычислялись из ключей или из агрегатных функций. То есть, каждый выбираемый из таблицы столбец, должен использоваться либо в ключах, либо внутри агрегатных функций. Чтобы получить поведение, как в MySQL, вы можете поместить остальные столбцы в агрегатную функцию `any`.

## stddevPop

Результат равен квадратному корню от `varPop(x)`.

### Примечание

Функция использует вычислительно неустойчивый алгоритм. Если для ваших расчётов необходима **вычислительная устойчивость**, используйте функцию `stddevPopStable`. Она работает медленнее, но обеспечивает меньшую вычислительную ошибку.

## stddevSamp

Результат равен квадратному корню от `varSamp(x)`.

### Примечание

Функция использует вычислительно неустойчивый алгоритм. Если для ваших расчётов необходима **вычислительная устойчивость**, используйте функцию `stddevSampStable`. Она работает медленнее, но обеспечивает меньшую вычислительную ошибку.

## varPop(x)

Вычисляет величину  $\sum((x - \bar{x})^2) / n$ , где  $n$  - размер выборки,  $\bar{x}$ - среднее значение  $x$ .

То есть, дисперсию для множества значений. Возвращает `Float64`.

### Примечание

Функция использует вычислительно неустойчивый алгоритм. Если для ваших расчётов необходима **вычислительная устойчивость**, используйте функцию `varPopStable`. Она работает медленнее, но обеспечивает меньшую вычислительную ошибку.

## varSamp

Вычисляет величину  $\sum((x - \bar{x})^2) / (n - 1)$ , где  $n$  - размер выборки,  $\bar{x}$ - среднее значение  $x$ .

Она представляет собой несмешённую оценку дисперсии случайной величины, если переданные в функцию значения являются выборкой этой случайной величины.

Возвращает `Float64`. В случае, когда  $n \leq 1$ , возвращается  $+\infty$ .

### Примечание

Функция использует вычислительно неустойчивый алгоритм. Если для ваших расчётов необходима **вычислительная устойчивость**, используйте функцию `varSampStable`. Она работает медленнее, но обеспечивает меньшую вычислительную ошибку.

## covarPop

Синтаксис: `covarPop(x, y)`

Вычисляет величину  $\sum((x - \bar{x})(y - \bar{y})) / n$ .

### Примечание

Функция использует вычислительно неустойчивый алгоритм. Если для ваших расчётов необходима **вычислительная устойчивость**, используйте функцию `covarPopStable`. Она работает медленнее, но обеспечивает меньшую вычислительную ошибку.

## Перечень агрегатных функций

Стандартные агрегатные функции:

- `count`
- `min`
- `max`
- `sum`
- `avg`
- `any`
- `stddevPop`
- `stddevSamp`
- `varPop`
- `varSamp`
- `covarPop`
- `covarSamp`

Агрегатные функции, специфичные для ClickHouse:

- `anyHeavy`
- `anyLast`
- `argMin`
- `argMax`
- `avgWeighted`
- `topK`
- `topKWeighted`
- `groupArray`
- `groupUniqArray`
- `groupArrayInsertAt`
- `groupArrayMovingAvg`
- `groupArrayMovingSum`
- `groupBitAnd`
- `groupBitOr`
- `groupBitXor`
- `groupBitmap`
- `sumWithOverflow`
- `sumMap`
- `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](#)

---

## covarSamp

Синтаксис: `covarSamp(x, y)`

Вычисляет величину  $\Sigma((x - \bar{x})(y - \bar{y})) / (n - 1)$

Возвращает `Float64`. В случае, когда `n <= 1`, возвращается  $+\infty$ .

### Примечание

Функция использует вычислительно неустойчивый алгоритм. Если для ваших расчётов необходима **вычислительная устойчивость**, используйте функцию [covarSampStable](#). Она работает медленнее, но обеспечивает меньшую вычислительную ошибку.

## anyHeavy

Выбирает часто встречающееся значение с помощью алгоритма «heavy hitters». Если существует значение, которое встречается чаще, чем в половине случаев, в каждом потоке выполнения запроса, то возвращается данное значение. В общем случае, результат недетерминирован.

```
anyHeavy(column)
```

### Аргументы

- `column` — имя столбца.

### Пример

Возьмём набор данных `OnTime` и выберем произвольное часто встречающееся значение в столбце `AirlineID`.

```
SELECT anyHeavy(AirlineID) AS res  
FROM ontime
```

```
res  
19690 |
```

## anyLast

Выбирает последнее попавшееся значение.

Результат так же недетерминирован, как и для функции `any`.

## argMin

Вычисляет значение `arg` при минимальном значении `val`. Если есть несколько разных значений `arg` для минимальных значений `val`, возвращает первое попавшееся из таких значений.

### Синтаксис

```
argMin(arg, val)
```

### Аргументы

- `arg` — аргумент.
- `val` — значение.

### Возвращаемое значение

- Значение `arg`, соответствующее минимальному значению `val`.

Тип: соответствует типу `arg`.

### Пример

Исходная таблица:

user	salary
director	5000
manager	3000
worker	1000

Запрос:

```
SELECT argMin(user, salary) FROM salary;
```

Результат:

argMin(user, salary)
worker

## argMax

Вычисляет значение `arg` при максимальном значении `val`. Если есть несколько разных значений `arg` для максимальных значений `val`, возвращает первое попавшееся из таких значений.

### Синтаксис

```
argMax(arg, val)
```

### Аргументы

- `arg` — аргумент.
- `val` — значение.

### Возвращаемое значение

- значение `arg`, соответствующее максимальному значению `val`.

Тип: соответствует типу `arg`.

### Пример

Исходная таблица:

user	salary
director	5000
manager	3000
worker	1000

Запрос:

```
SELECT argMax(user, salary), argMax(tuple(user, salary), salary) FROM salary;
```

Результат:

```
argMax(user, salary) ┌─ argMax(tuple(user, salary), salary) ─┐  
director      | ('director', 5000)   |
```

## avgWeighted

Вычисляет **среднее арифметическое взвешенное**.

### Синтаксис

```
avgWeighted(x, weight)
```

### Аргументы

- `x` — значения. **Целые числа** или **числа с плавающей запятой**.
- `weight` — веса отдельных значений. **Целые числа** или **числа с плавающей запятой**.

Типы параметров должны совпадать.

### Возвращаемое значение

- Среднее арифметическое взвешенное.
- `Nan`, если все веса равны 0.

Тип: `Float64`

### Пример

Запрос:

```
SELECT avgWeighted(x, w)  
FROM values('x Int8, w Int8', (4, 1), (1, 0), (10, 2))
```

Результат:

```
avgWeighted(x, weight) ┌─  
8 |
```

## corr

Синтаксис: `corr(x, y)`

Вычисляет коэффициент корреляции Пирсона:  $\frac{\sum((x - \bar{x})(y - \bar{y}))}{\sqrt{\sum((x - \bar{x})^2) * \sum((y - \bar{y})^2)}}$

### Примечание

Функция использует вычислительно неустойчивый алгоритм. Если для ваших расчётов необходима **вычислительная устойчивость**, используйте функцию `corrStable`. Она работает медленнее, но обеспечивает меньшую вычислительную ошибку.

# topK

Возвращает массив наиболее часто встречающихся значений в указанном столбце.

Результирующий массив упорядочен по убыванию частоты значения (не по самим значениям).

Реализует [Filtered Space-Saving](#) алгоритм для анализа ТопК, на основе reduce-and-combine алгоритма из методики [Parallel Space Saving](#).

```
topK(N)(column)
```

Функция не дает гарантированного результата. В некоторых ситуациях могут возникать ошибки, и функция возвращает частые, но не наиболее частые значения.

Рекомендуем использовать значения  $N < 10$ , при больших  $N$  снижается производительность.

Максимально возможное значение  $N = 65536$ .

## Аргументы

- $N$  — количество значений.
- $x$  — столбец.

## Пример

Возьмём набор данных [OnTime](#) и выберем 3 наиболее часто встречающихся значения в столбце `AirlineID`.

```
SELECT topK(3)(AirlineID) AS res  
FROM ontime
```

```
res  
[19393,19790,19805] |
```

# topKWeighted

Возвращает массив наиболее часто встречающихся значений в указанном столбце.

Результирующий массив упорядочен по убыванию частоты значения (не по самим значениям).

Дополнительно учитывается вес значения.

## Синтаксис

```
topKWeighted(N)(x, weight)
```

## Аргументы

- $N$  — количество элементов для выдачи.
- $x$  — значение.
- $weight$  — вес. Каждое значение учитывается  $weight$  раз при расчёте частоты. [UInt64](#).

## Возвращаемое значение

Возвращает массив значений с максимально приближенной суммой весов.

## Пример

Запрос:

```
SELECT topKWeighted(10)(number, number) FROM numbers(1000)
```

Результат:

```
topKWeighted(10)(number, number)
[999,998,997,996,995,994,993,992,991,990] |
```

## Смотрите также

- [topK](#)

# groupArray

Синтаксис: `groupArray(x)` или `groupArray(max_size)(x)`

Составляет массив из значений аргумента.

Значения в массив могут быть добавлены в любом (недетерминированном) порядке.

Вторая версия (с параметром `max_size`) ограничивает размер результирующего массива `max_size` элементами.

Например, `groupArray(1)(x)` эквивалентно `[any(x)]`.

В некоторых случаях, вы всё же можете рассчитывать на порядок выполнения запроса. Это — случаи, когда `SELECT` идёт из подзапроса, в котором используется `ORDER BY`.

# groupUniqArray

Синтаксис: `groupUniqArray(x)` или `groupUniqArray(max_size)(x)`

Составляет массив из различных значений аргумента. Расход оперативной памяти такой же, как у функции `uniqExact`.

Функция `groupUniqArray(max_size)(x)` ограничивает размер результирующего массива до `max_size` элементов. Например, `groupUniqArray(1)(x)` равнозначно `[any(x)]`.

# groupArrayInsertAt

Вставляет значение в заданную позицию массива.

## Синтаксис

```
groupArrayInsertAt(default_x, size)(x, pos)
```

Если запрос вставляет вставляется несколько значений в одну и ту же позицию, то функция ведет себя следующим образом:

- Если запрос выполняется в одном потоке, то используется первое из вставляемых значений.
- Если запрос выполняется в нескольких потоках, то в результирующем массиве может оказаться любое из вставляемых значений.

## Аргументы

- `x` — значение, которое будет вставлено. **Выражение**, возвращающее значение одного из **поддерживаемых типов данных**.
- `pos` — позиция, в которую вставляется заданный элемент `x`. Нумерация индексов в массиве начинается с нуля. **UInt32**.
- `default_x` — значение по умолчанию для подстановки на пустые позиции. Опциональный параметр. **Выражение**, возвращающее значение с типом параметра `x`. Если `default_x` не определен, используются **значения по умолчанию**.
- `size` — длина результирующего массива. Опциональный параметр. При использовании этого параметра должно быть указано значение по умолчанию `default_x`. **UInt32**.

## Возвращаемое значение

- Массив со вставленными значениями.

Тип: **Array**.

## Примеры

Запрос:

```
SELECT groupArrayInsertAt(toString(number), number * 2) FROM numbers(5);
```

Результат:

```
groupArrayInsertAt(toString(number), multiply(number, 2))—  
['0','1','2','3','4'] |
```

Запрос:

```
SELECT groupArrayInsertAt('-')(toString(number), number * 2) FROM numbers(5);
```

Результат:

```
groupArrayInsertAt('')(toString(number), multiply(number, 2))—  
['0','-','1','-','2','-','3','-','4'] |
```

Запрос:

```
SELECT groupArrayInsertAt('-', 5)(toString(number), number * 2) FROM numbers(5);
```

Результат:

```
groupArrayInsertAt('-', 5)(toString(number), multiply(number, 2))—  
['0','-', '1', ' ', '2'] |
```

Многопоточная вставка элементов в одну позицию.

Запрос:

```
SELECT groupArrayInsertAt(number, 0) FROM numbers_mt(10) SETTINGS max_block_size = 1;
```

В результат этого запроса мы получите случайное целое число в диапазоне [0,9]. Например:

```
groupArrayInsertAt(number, 0)  
[7]
```

## groupArrayMovingSum

Вычисляет скользящую сумму входных значений.

```
groupArrayMovingSum(numbers_for_summing)  
groupArrayMovingSum(window_size)(numbers_for_summing)
```

Функция может принимать размер окна в качестве параметра. Если окно не указано, то функция использует размер окна, равный количеству строк в столбце.

### Аргументы

- `numbers_for_summing` — выражение, возвращающее значение числового типа.
- `window_size` — размер окна.

### Возвращаемые значения

- Массив того же размера и типа, что и входные данные.

### Пример

Таблица с исходными данными:

```
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

Запросы:

```
SELECT  
    groupArrayMovingSum(int) AS I,  
    groupArrayMovingSum(float) AS F,  
    groupArrayMovingSum(dec) AS D  
FROM t
```

```
[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
```

```
[1,3,6,11] | [1.1,3.3000002,6.6000004,12.17] | [1.10,3.30,6.60,12.17]
```

## groupArrayMovingAvg

Вычисляет скользящее среднее для входных значений.

```
groupArrayMovingAvg(numbers_for_summing)
groupArrayMovingAvg(window_size)(numbers_for_summing)
```

Функция может принимать размер окна в качестве параметра. Если окно не указано, то функция использует размер окна, равный количеству строк в столбце.

### Аргументы

- `numbers_for_summing` — выражение, возвращающее значение числового типа.
- `window_size` — размер окна.

### Возвращаемые значения

- Массив того же размера и типа, что и входные данные.

Функция использует округление к меньшему по модулю. Оно усекает десятичные разряды, незначимые для результирующего типа данных.

### Пример

Таблица с исходными данными:

```
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

Запросы:

```
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

Создает массив из случайно выбранных значений аргумента. Количество элементов в массиве ограничено значением параметра `max_size`. Элементы добавляются в результирующий массив в случайном порядке.

### Синтаксис

```
groupArraySample(max_size[, seed])(x)
```

### Аргументы

- `max_size` — максимальное количество элементов в возвращаемом массиве. **UInt64**.
- `seed` — состояние генератора случайных чисел. Необязательный параметр. **UInt64**. Значение по умолчанию: 123456.
- `x` — аргумент (название колонки таблицы или выражение).

### Возвращаемые значения

- Массив случайно выбранных значений аргумента `x`.

Тип: **Массив**.

### Примеры

Рассмотрим таблицу `colors`:

id	color
1	red
2	blue
3	green
4	white
5	orange

Запрос с названием колонки таблицы в качестве аргумента:

```
SELECT groupArraySample(3)(color) as newcolors FROM colors;
```

Результат:

```
newcolors  
['white','blue','green'] |
```

Запрос с названием колонки и другим состоянием генератора случайных чисел:

```
SELECT groupArraySample(3, 987654321)(color) as newcolors FROM colors;
```

Результат:

```
newcolors  
['red','orange','green'] |
```

Запрос с выражением в качестве аргумента:

```
SELECT groupArraySample(3)(concat('light-', color)) as newcolors FROM colors;
```

Результат:

```
newcolors  
['light-blue','light-orange','light-green'] |
```

## groupBitAnd

Применяет побитовое И для последовательности чисел.

```
groupBitAnd(expr)
```

### Аргументы

expr – выражение, результат которого имеет тип данных UInt\*.

### Возвращаемое значение

Значение типа UInt\*.

### Пример

Тестовые данные:

```
binary    decimal
00101100 = 44
00011100 = 28
00001101 = 13
01010101 = 85
```

Запрос:

```
SELECT groupBitAnd(num) FROM t
```

Где num — столбец с тестовыми данными.

Результат:

```
binary    decimal  
00000100 = 4
```

## groupBitOr

Применяет побитовое ИЛИ для последовательности чисел.

```
groupBitOr(expr)
```

### Аргументы

expr – выражение, результат которого имеет тип данных UInt\*.

### Возвращаемое значение

Значение типа UInt\*.

### Пример

Тестовые данные:

```
binary    decimal  
00101100 = 44  
00011100 = 28  
00001101 = 13  
01010101 = 85
```

Запрос:

```
SELECT groupBitOr(num) FROM t
```

Где num — столбец с тестовыми данными.

Результат:

```
binary    decimal  
01111101 = 125
```

## groupBitXor

Применяет побитовое ИСКЛЮЧАЮЩЕЕ ИЛИ для последовательности чисел.

```
groupBitXor(expr)
```

### Аргументы

`expr` – выражение, результат которого имеет тип данных `UInt*`.

### **Возвращаемое значение**

Значение типа `UInt`.

### **Пример**

Тестовые данные:

```
binary    decimal
00101100 = 44
00011100 = 28
00001101 = 13
01010101 = 85
```

Запрос:

```
SELECT groupBitXor(num) FROM t
```

Где `num` — столбец с тестовыми данными.

Результат:

```
binary    decimal
01101000 = 104
```

## groupBitmap

Bitmap или агрегатные вычисления для столбца с типом данных `UInt*`, возвращают кардинальность в виде значения типа `UInt64`, если добавить суффикс `-State`, то возвращают [объект bitmap](#).

```
groupBitmap(expr)
```

### **Аргументы**

`expr` – выражение, результат которого имеет тип данных `UInt*`.

### **Возвращаемое значение**

Значение типа `UInt64`.

### **Пример**

Тестовые данные:

```
UserID
1
1
2
2
3
```

Запрос:

```
SELECT groupBitmap(UserID) as num FROM t
```

Результат:

```
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

Вычисляет сумму чисел, используя для результата тот же тип данных, что и для входных параметров. Если сумма выйдет за максимальное значение для заданного типа данных, то функция вернёт ошибку.

Работает только для чисел.

## deltaSum

Суммирует арифметическую разницу между последовательными строками. Если разница отрицательна — она будет проигнорирована.

### Примечание

Чтобы эта функция работала должным образом, исходные данные должны быть отсортированы. В [материализованном представлении](#) вместо нее рекомендуется использовать [deltaSumTimestamp](#).

### Синтаксис

```
deltaSum(value)
```

### Аргументы

- `value` — входные значения, должны быть типа [Integer](#) или [Float](#).

### Возвращаемое значение

- Накопленная арифметическая разница.

Тип: [Integer](#) или [Float](#).

### Примеры

Запрос:

```
SELECT deltaSum(arrayJoin([1, 2, 3]));
```

Результат:

```
deltaSum(arrayJoin([1, 2, 3]))  
2 |
```

Запрос:

```
SELECT deltaSum(arrayJoin([1, 2, 3, 0, 3, 4, 2, 3]));
```

Результат:

```
deltaSum(arrayJoin([1, 2, 3, 0, 3, 4, 2, 3]))  
7 |
```

Запрос:

```
SELECT deltaSum(arrayJoin([2.25, 3, 4.5]));
```

Результат:

```
deltaSum(arrayJoin([2.25, 3, 4.5]))  
2.25 |
```

## Смотрите также

- [runningDifference](#)

## deltaSumTimestamp

Суммирует разницу между последовательными строками. Если разница отрицательна — она будет проигнорирована.

Эта функция предназначена в первую очередь для [материализованных представлений](#), упорядоченных по некоторому временному бакету согласно timestamp, например, по бакету `toStartOfMinute`. Поскольку строки в таком материализованном представлении будут иметь одинаковый timestamp, невозможно объединить их в "правом" порядке. Функция отслеживает timestamp наблюдаемых значений, поэтому возможно правильно упорядочить состояния во время слияния.

Чтобы вычислить разницу между упорядоченными последовательными строками, вы можете использовать функцию `deltaSum` вместо функции `deltaSumTimestamp`.

### Синтаксис

```
deltaSumTimestamp(value, timestamp)
```

## Аргументы

- `value` — входные значения, должны быть типа `Integer`, или `Float`, или `Date`, или `DateTime`.
- `timestamp` — параметр для упорядочивания значений, должен быть типа `Integer`, или `Float`, или `Date`, или `DateTime`.

## Возвращаемое значение

- Накопленная разница между последовательными значениями, упорядоченными по параметру `timestamp`.

Тип: `Integer`, или `Float`, или `Date`, или `DateTime`.

## Пример

Запрос:

```
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));
```

Результат:

```
deltaSumTimestamp(value, timestamp)─
 13 |
```

## sumMap(key, value), sumMap(Tuple(key, value))

Производит суммирование массива 'value' по соответствующим ключам заданным в массиве 'key'. Передача кортежа ключей и значений массива синонимично передаче двух массивов ключей и значений.

Количество элементов в 'key' и 'value' должно быть одинаковым для каждой строки, для которой происходит суммирование.

Возвращает кортеж из двух массивов - ключи в отсортированном порядке и значения, просуммированные по соответствующим ключам.

## Пример:

```
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

Синтаксис: `minMap(key, value)` or `minMap(Tuple(key, value))`

Вычисляет минимальное значение массива `value` в соответствии с ключами, указанными в массиве `key`.

Передача кортежа ключей и массивов значений идентична передаче двух массивов ключей и значений.

Количество элементов в параметрах `key` и `value` должно быть одинаковым для каждой суммируемой строки.

Возвращает кортеж из двух массивов: ключи в отсортированном порядке и значения, рассчитанные для соответствующих ключей.

Пример:

```
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

Синтаксис: `maxMap(key, value)` or `maxMap(Tuple(key, value))`

Вычисляет максимальные значения массива `value`, соответствующие ключам, указанным в массиве `key`.

Передача кортежа ключей и массивов значений идентична передаче двух массивов ключей и значений.

Количество элементов в параметрах `key` и `value` должно быть одинаковым для каждой суммируемой строки.

Возвращает кортеж из двух массивов: ключи и значения, рассчитанные для соответствующих ключей.

Пример:

```
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

Вычисляет сумму чисел и одновременно подсчитывает количество строк.

### Синтаксис

```
sumCount(x)
```

### Аргументы

- `x` — Входное значение типа [Integer](#), [Float](#), или [Decimal](#).

### Возвращаемое значение

- Кортеж из элементов (`sum`, `count`), где `sum` — это сумма чисел и `count` — количество строк со значениями, отличными от `NULL`.

Тип: [Tuple](#).

### Пример

Запрос:

```
CREATE TABLE s_table (x Nullable(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;
```

Результат:

```
sumCount(x)
(190,20) |
```

### Смотрите также

- Настройка [optimize\\_syntax\\_fuse\\_functions](#)

## rankCorr

Вычисляет коэффициент ранговой корреляции.

### Синтаксис

```
rankCorr(x, y)
```

### Аргументы

- `x` — произвольное значение. [Float32](#) или [Float64](#).
- `y` — произвольное значение. [Float32](#) или [Float64](#).

## Возвращаемое значение

- Возвращает коэффициент ранговой корреляции рангов x и y. Значение коэффициента корреляции изменяется в пределах от -1 до +1. Если передается менее двух аргументов, функция возвращает исключение. Значение, близкое к +1, указывает на высокую линейную зависимость, и с увеличением одной случайной величины увеличивается вторая случайная величина. Значение, близкое к -1, указывает на высокую линейную зависимость, и с увеличением одной случайной величины вторая случайная величина уменьшается. Значение, близкое или равное 0, означает отсутствие связи между двумя случайными величинами.

Тип: [Float64](#).

## Пример

Запрос:

```
SELECT rankCorr(number, number) FROM numbers(100);
```

Результат:

```
rankCorr(number, number)─  
1 |
```

Запрос:

```
SELECT roundBankers(rankCorr(exp(number), sin(number)), 3) FROM numbers(100);
```

Результат:

```
roundBankers(rankCorr(exp(number), sin(number)), 3)─  
-0.037 |
```

## Смотрите также

- [Коэффициент ранговой корреляции Спирмена](#)

# sumKahan

Вычисляет сумму с использованием [компенсационного суммирования по алгоритму Кэхэна](#).

Работает медленнее функции [sum](#).

Компенсация работает только для [Float](#) типов.

## Синтаксис

```
sumKahan(x)
```

## Аргументы

- x — Входное значение типа [Integer](#), [Float](#), или [Decimal](#).

## Возвращаемое значение

- сумма чисел с типом [Integer](#), [Float](#), или [Decimal](#) зависящим от типа входящих аргументов

## Пример

Запрос:

```
SELECT sum(0.1), sumKahan(0.1) FROM numbers(10);
```

Результат:

sum(0.1)	sumKahan(0.1)
0.9999999999999999	1

## intervalLengthSum

Вычисляет длину объединения интервалов (отрезков на числовой оси).

### Синтаксис

```
intervalLengthSum(start, end)
```

### Аргументы

- `start` — начальное значение интервала. `Int32`, `Int64`, `UInt32`, `UInt64`, `Float32`, `Float64`, `DateTime` или `Date`.
- `end` — конечное значение интервала. `Int32`, `Int64`, `UInt32`, `UInt64`, `Float32`, `Float64`, `DateTime` или `Date`.

## Примечание

Аргументы должны быть одного типа. В противном случае ClickHouse сгенерирует исключение.

### Возвращаемое значение

- Длина объединения всех интервалов (отрезков на числовой оси). В зависимости от типа аргумента возвращаемое значение может быть типа `UInt64` или `Float64`.

### Примеры

1. Входная таблица:

id	start	end
a	1.1	2.9
a	2.5	3.2
a	4	5

В этом примере используются аргументы типа `Float32`. Функция возвращает значение типа `Float64`.

Результатом функции будет сумма длин интервалов  $[1.1, 3.2]$  (объединение  $[1.1, 2.9]$  и  $[2.5, 3.2]$ ) и  $[4, 5]$

Запрос:

```
SELECT id, intervalLengthSum(start, end), toTypeName(intervalLengthSum(start, end)) FROM fl_interval GROUP BY id
ORDER BY id;
```

Результат:

id	intervalLengthSum(start, end)	toTypeName(intervalLengthSum(start, end))
a	3.1	Float64

2. Входная таблица:

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

В этом примере используются аргументы типа DateTime. Функция возвращает значение, выраженное в секундах.

Запрос:

```
SELECT id, intervalLengthSum(start, end), toTypeName(intervalLengthSum(start, end)) FROM dt_interval GROUP BY id
ORDER BY id;
```

Результат:

id	intervalLengthSum(start, end)	toTypeName(intervalLengthSum(start, end))
a	6610	UInt64

3. Входная таблица:

id	start	end
a	2020-01-01	2020-01-04
a	2020-01-12	2020-01-18

В этом примере используются аргументы типа Date. Функция возвращает значение, выраженное в днях.

Запрос:

```
SELECT id, intervalLengthSum(start, end), toTypeName(intervalLengthSum(start, end)) FROM date_interval GROUP BY id
ORDER BY id;
```

Результат:

id	intervalLengthSum(start, end)	toTypeName(intervalLengthSum(start, end))
a	9	UInt64

Вычисляет **коэффициент асимметрии** для последовательности.

```
skewPop(expr)
```

## Аргументы

`expr` — **выражение**, возвращающее число.

## Возвращаемое значение

Коэффициент асимметрии заданного распределения. Тип — **Float64**

## Пример

```
SELECT skewPop(value) FROM series_with_value_column;
```

# skewSamp

Вычисляет **выборочный коэффициент асимметрии** для последовательности.

Он представляет собой несмешенную оценку асимметрии случайной величины, если переданные значения образуют ее выборку.

```
skewSamp(expr)
```

## Аргументы

`expr` — **выражение**, возвращающее число.

## Возвращаемое значение

Коэффициент асимметрии заданного распределения. Тип — **Float64**. Если  $n \leq 1$  ( $n$  — размер выборки), тогда функция возвращает `nan`.

## Пример

```
SELECT skewSamp(value) FROM series_with_value_column;
```

# kurtPop

Вычисляет **коэффициент эксцесса** последовательности.

```
kurtPop(expr)
```

## Аргументы

`expr` — **выражение**, возвращающее число.

## Возвращаемое значение

Коэффициент эксцесса заданного распределения. Тип — **Float64**

## Пример

```
SELECT kurtPop(value) FROM series_with_value_column;
```

## kurtSamp

Вычисляет **выборочный коэффициент эксцесса** для последовательности.

Он представляет собой несмешенную оценку эксцесса случайной величины, если переданные значения образуют ее выборку.

```
kurtSamp(expr)
```

### Аргументы

`expr` — **выражение**, возвращающее число.

### Возвращаемое значение

Коэффициент эксцесса заданного распределения. Тип — **Float64**. Если  $n \leq 1$  ( $n$  — размер выборки), тогда функция возвращает `nan`.

### Пример

```
SELECT kurtSamp(value) FROM series_with_value_column;
```

## uniq

Приближённо вычисляет количество различных значений аргумента.

```
uniq(x[, ...])
```

### Аргументы

Функция принимает переменное число входных параметров. Параметры могут быть числовых типов, а также `Tuple`, `Array`, `Date`, `DateTime`, `String`.

### Возвращаемое значение

- Значение с типом данных **UInt64**.

### Детали реализации

Функция:

- Вычисляет хэш для всех параметров агрегации, а затем использует его в вычислениях.
- Использует адаптивный алгоритм выборки. В качестве состояния вычисления функция использует выборку хэш-значений элементов размером до 65536.

Этот алгоритм очень точен и очень эффективен по использованию CPU. Если запрос содержит небольшое количество этих функций, использование `uniq` почти так же эффективно, как и использование других агрегатных функций.

- Результат детерминирован (не зависит от порядка выполнения запроса).

Эту функцию рекомендуется использовать практически во всех сценариях.

## Смотрите также

- [uniqCombined](#)
- [uniqCombined64](#)
- [uniqHLL12](#)
- [uniqExact](#)

## uniqExact

Вычисляет точное количество различных значений аргументов.

```
uniqExact(x[, ...])
```

Функцию `uniqExact` следует использовать, если вам обязательно нужен точный результат. В противном случае используйте функцию [uniq](#).

Функция `uniqExact` расходует больше оперативной памяти, чем функция `uniq`, так как размер состояния неограниченно растёт по мере роста количества различных значений.

## Аргументы

Функция принимает переменное число входных параметров. Параметры могут быть числовых типов, а также `Tuple`, `Array`, `Date`, `DateTime`, `String`.

## Смотрите также

- [uniq](#)
- [uniqCombined](#)
- [uniqHLL12](#)

## uniqCombined

Приближённо вычисляет количество различных значений аргумента.

```
uniqCombined(HLL_precision)(x[, ...])
```

Функция `uniqCombined` — это хороший выбор для вычисления количества различных значений.

## Аргументы

Функция принимает переменное число входных параметров. Параметры могут быть числовых типов, а также `Tuple`, `Array`, `Date`, `DateTime`, `String`.

`HLL_precision` — это логарифм по основанию 2 от числа ячеек в [HyperLogLog](#). Необязательный, можно использовать функцию как `uniqCombined(x [...])`. Для `HLL_precision` значение по умолчанию — 17, что фактически составляет 96 КБ пространства ( $2^{17}$  ячеек, 6 бит каждая).

## Возвращаемое значение

- Число типа [UInt64](#).

## Детали реализации

Функция:

- Вычисляет хэш (64-битный для `String` и 32-битный для всех остальных типов) для всех параметров агрегации, а затем использует его в вычислениях.
- Используется комбинация трёх алгоритмов: массив, хэш-таблица и HyperLogLog с таблицей коррекции погрешности.

Для небольшого количества различных значений используется массив. Если размер набора больше, используется хэш-таблица. При дальнейшем увеличении количества значений, используется структура HyperLogLog, имеющая фиксированный размер в памяти.

- Результат детерминирован (не зависит от порядка выполнения запроса).

## Note

Так как используется 32-битный хэш для не-`String` типов, результат будет иметь очень очень большую ошибку для количества различных элементов существенно больше `UINT_MAX` (ошибка быстро растёт начиная с нескольких десятков миллиардов различных значений), таким образом в этом случае нужно использовать **uniqCombined64**

По сравнению с функцией `uniq`, `uniqCombined`:

- Потребляет в несколько раз меньше памяти.
- Вычисляет с в несколько раз более высокой точностью.
- Обычно имеет немного более низкую производительность. В некоторых сценариях `uniqCombined` может показывать более высокую производительность, чем `uniq`, например, в случае распределенных запросов, при которых по сети передаётся большое количество состояний агрегации.

## Смотрите также

- [uniq](#)
- [uniqCombined64](#)
- [uniqHLL12](#)
- [uniqExact](#)

## uniqCombined64

Использует 64-битный хэш для всех типов, в отличие от `uniqCombined`.

## uniqHLL12

Вычисляет приблизительное число различных значений аргументов, используя алгоритм `HyperLogLog`.

`uniqHLL12(x[, ...])`

## Аргументы

Функция принимает переменное число входных параметров. Параметры могут быть числовых типов, а также `Tuple`, `Array`, `Date`, `DateTime`, `String`.

## Возвращаемое значение

- Значение хэша с типом данных `UInt64`.

## Детали реализации

Функция:

- Вычисляет хэш для всех параметров агрегации, а затем использует его в вычислениях.
- Использует алгоритм HyperLogLog для аппроксимации числа различных значений аргументов.

Используется  $2^{12}$  5-битовых ячеек. Размер состояния чуть больше 2.5 КБ. Результат не точный (ошибка до ~10%) для небольших множеств (<10К элементов). Однако для множеств большой кардинальности (10К - 100М) результат довольно точен (ошибка до ~1.6%). Начиная с 100М ошибка оценки будет только расти и для множеств огромной кардинальности (1B+ элементов) функция возвращает результат с очень большой неточностью.

- Результат детерминирован (не зависит от порядка выполнения запроса).

Мы не рекомендуем использовать эту функцию. В большинстве случаев используйте функцию `uniq` или `uniqCombined`.

- `uniq`
- `uniqCombined`
- `uniqExact`

## 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

Приблизительно вычисляет [квантиль](#) числовой последовательности.

Функция использует алгоритм [reservoir sampling](#) с размером резервуара до 8192 и случайным генератором чисел для сэмплирования. Результат не детерминирован. Чтобы получить точную квантиль используйте функцию [quantileExact](#).

Внутренние состояния функций `quantile*` не объединяются, если они используются в одном запросе. Если вам необходимо вычислить квантили нескольких уровней, используйте функцию [quantiles](#), это повысит эффективность запроса.

## Синтаксис

```
quantile(level)(expr)
```

Алиас: `median`.

## Аргументы

- `level` — уровень квантили. Опционально. Константное значение с плавающей запятой от 0 до 1. Мы рекомендуем использовать значение `level` из диапазона [0.01, 0.99]. Значение по умолчанию: 0.5. При `level=0.5` функция вычисляет [медиану](#).
- `expr` — выражение, зависящее от значений столбцов, возвращающее данные [числовых типов](#) или типов [Date](#), [DateTime](#).

## Возвращаемое значение

- Приблизительный квантиль заданного уровня.

Тип:

- [Float64](#) для входных данных числового типа.
- [Date](#), если входные значения имеют тип `Date`.
- [DateTime](#), если входные значения имеют тип `DateTime`.

## Пример

Входная таблица:

val
1
1
2
3

Запрос:

```
SELECT quantile(val) FROM t
```

Результат:

quantile(val)
1.5

## Смотрите также

- [median](#)
- [quantiles](#)

# ФУНКЦИИ ДЛЯ НЕСКОЛЬКИХ КВАНТИЛЕЙ

## quantiles

Синтаксис: `quantiles(level1, level2, ...)(x)`

Все функции для вычисления квантилей имеют соответствующие функции для вычисления нескольких квантилей: `quantiles`, `quantilesDeterministic`, `quantilesTiming`, `quantilesTimingWeighted`, `quantilesExact`, `quantilesExactWeighted`, `quantilesTDigest`, `quantilesBFloat16`. Эти функции вычисляют все квантили указанных уровней в один проход и возвращают массив с вычисленными значениями.

## quantilesExactExclusive

Точно вычисляет [квантили](#) числовой последовательности.

Чтобы получить точный результат, все переданные значения собираются в массив, который затем частично сортируется. Таким образом, функция потребляет объем памяти  $O(n)$ , где  $n$  — количество переданных значений. Для небольшого числа значений эта функция эффективна.

Эта функция эквивалентна Excel функции [PERCENTILE.EXC](#), тип R6.

С наборами уровней работает эффективнее, чем [quantileExactExclusive](#).

### Синтаксис

```
quantilesExactExclusive(level1, level2, ...)(expr)
```

### Аргументы

- `expr` — выражение, зависящее от значений столбцов. Возвращает данные [числовых типов](#), [Date](#) или [DateTime](#).

### Параметры

- `level` — уровень квантилей. Возможные значения: (0, 1) — граничные значения не учитываются. [Float](#).

## Возвращаемые значения

- **Массив** квантилей указанных уровней.

Тип значений массива:

- [Float64](#) для входных данных числового типа.
- [Date](#), если входные значения имеют тип [Date](#).
- [DateTime](#), если входные значения имеют тип [DateTime](#).

## Пример

Запрос:

```
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);
```

Результат:

```
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

Точно вычисляет [квантили](#) числовой последовательности.

Чтобы получить точный результат, все переданные значения собираются в массив, который затем частично сортируется. Таким образом, функция потребляет объем памяти  $O(n)$ , где  $n$  — количество переданных значений. Для небольшого числа значений эта функция эффективна.

Эта функция эквивалентна Excel функции [PERCENTILE.INC](#), тип [R7](#).

С наборами уровней работает эффективнее, чем [quantileExactInclusive](#).

## Синтаксис

```
quantilesExactInclusive(level1, level2, ...)(expr)
```

## Аргументы

- `expr` — выражение, зависящее от значений столбцов. Возвращает данные [числовых типов](#), [Date](#) или [DateTime](#).

## Параметры

- `level` — уровень квантилей. Возможные значения: [0, 1] — граничные значения учитываются. [Float](#).

## Возвращаемые значения

- **Массив** квантилей указанных уровней.

Тип значений массива:

- **Float64** для входных данных числового типа.
- **Date**, если входные значения имеют тип `Date`.
- **DateTime**, если входные значения имеют тип `DateTime`.

## Пример

Запрос:

```
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);
```

Результат:

```
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]
```

# ФУНКЦИИ `quantileExact`

## `quantileExact`

Точно вычисляет **квантиль** числовой последовательности.

Чтобы получить точный результат, все переданные значения собираются в массив, который затем частично сортируется. Таким образом, функция потребляет объем памяти  $O(n)$ , где  $n$  — количество переданных значений. Для небольшого числа значений эта функция эффективна.

Внутренние состояния функций `quantile*` не объединяются, если они используются в одном запросе. Если вам необходимо вычислить квантили нескольких уровней, используйте функцию `quantiles`, это повысит эффективность запроса.

### Синтаксис

```
quantileExact(level)(expr)
```

Алиас: `medianExact`.

### Аргументы

- `level` — уровень квантили. Опционально. Константное значение с плавающей запятой от 0 до 1. Мы рекомендуем использовать значение `level` из диапазона `[0.01, 0.99]`. Значение по умолчанию: 0.5. При `level=0.5` функция вычисляет **медиану**.
- `expr` — выражение, зависящее от значений столбцов, возвращающее данные **числовых типов** или типов `Date`, `DateTime`.

### Возвращаемое значение

- Квантиль заданного уровня.

Тип:

- **Float64** для входных данных числового типа.
- **Date**, если входные значения имеют тип `Date`.

- **DateTime**, если входные значения имеют тип **DateTime**.

## Пример

Запрос:

```
SELECT quantileExact(number) FROM numbers(10)
```

Результат:

```
quantileExact(number)─  
5 |
```

## quantileExactLow

Как и `quantileExact`, эта функция вычисляет точный **квантиль** числовой последовательности данных.

Чтобы получить точное значение, все переданные значения объединяются в массив, который затем полностью сортируется. Сложность **алгоритма сортировки** равна  $O(N \cdot \log(N))$ , где  $N = \text{std}::\text{distance}(\text{first}, \text{last})$ .

Возвращаемое значение зависит от уровня квантили и количества элементов в выборке, то есть если уровень 0,5, то функция возвращает нижнюю медиану при чётном количестве элементов и медиану при нечётном. Медиана вычисляется аналогично реализации `median_low`, которая используется в python.

Для всех остальных уровней возвращается элемент с индексом, соответствующим значению `level * size_of_array`. Например:

```
SELECT quantileExactLow(0.1)(number) FROM numbers(10)
```

```
quantileExactLow(0.1)(number)─  
1 |
```

При использовании в запросе нескольких функций `quantile*` с разными уровнями, внутренние состояния не объединяются (то есть запрос работает менее эффективно). В этом случае используйте функцию **quantiles**.

## Синтаксис

```
quantileExact(level)(expr)
```

Аlias: `medianExactLow`.

## Аргументы

- `level` — уровень квантили. Опциональный параметр. Константное значение с плавающей запятой от 0 до 1. Мы рекомендуем использовать значение `level` из диапазона [0.01, 0.99]. Значение по умолчанию: 0.5. При `level=0.5` функция вычисляет **медиану**.
- `expr` — выражение, зависящее от значений столбцов, возвращающее данные **числовых типов**, **Date** или **DateTime**.

## Возвращаемое значение

- Квантиль заданного уровня.

Тип:

- **Float64** для входных данных числового типа.
- **Date** если входные значения имеют тип Date.
- **DateTime** если входные значения имеют тип DateTime.

## Пример

Запрос:

```
SELECT quantileExactLow(number) FROM numbers(10)
```

Результат:

```
└─quantileExactLow(number)─  
    4 |
```

## quantileExactHigh

Как и `quantileExact`, эта функция вычисляет точный **квантиль** числовой последовательности данных.

Все переданные значения объединяются в массив, который затем сортируется, чтобы получить точное значение. Сложность **алгоритма сортировки** равна  $O(N \cdot \log(N))$ , где  $N = \text{std::distance(first, last)}$ .

Возвращаемое значение зависит от уровня квантили и количества элементов в выборке, то есть если уровень 0,5, то функция возвращает верхнюю медиану при чётном количестве элементов и медиану при нечётном. Медиана вычисляется аналогично реализации **median\_high**, которая используется в python. Для всех остальных уровней возвращается элемент с индексом, соответствующим значению `level * size_of_array`.

Эта реализация ведет себя точно так же, как `quantileExact`.

При использовании в запросе нескольких функций `quantile*` с разными уровнями, внутренние состояния не объединяются (то есть запрос работает менее эффективно). В этом случае используйте функцию **quantiles**.

## Синтаксис

```
quantileExactHigh(level)(expr)
```

Алиас: `medianExactHigh`.

## Аргументы

- `level` — уровень квантили. Опциональный параметр. Константное значение с плавающей запятой от 0 до 1. Мы рекомендуем использовать значение `level` из диапазона [0.01, 0.99]. Значение по умолчанию: 0.5. При `level=0.5` функция вычисляет **медиану**.
- `expr` — выражение, зависящее от значений столбцов, возвращающее данные **числовых типов**, **Date** или **DateTime**.

## Возвращаемое значение

- Квантиль заданного уровня.

Тип:

- **Float64** для входных данных числового типа.
- **Date** если входные значения имеют тип **Date**.
- **DateTime** если входные значения имеют тип **DateTime**.

## Пример

Запрос:

```
SELECT quantileExactHigh(number) FROM numbers(10)
```

Результат:

```
quantileExactHigh(number)  
5 |
```

## quantileExactExclusive

Точно вычисляет **квантиль** числовой последовательности.

Чтобы получить точный результат, все переданные значения собираются в массив, который затем частично сортируется. Таким образом, функция потребляет объем памяти  $O(n)$ , где  $n$  — количество переданных значений. Для небольшого числа значений эта функция эффективна.

Эта функция эквивалентна Excel функции **PERCENTILE.EXC**, тип R6.

Если в одном запросе вызывается несколько функций **quantileExactExclusive** с разными значениями **level**, эти функции вычисляются независимо друг от друга. В таких случаях используйте функцию **quantilesExactExclusive**, запрос будет выполняться эффективнее.

## Синтаксис

```
quantileExactExclusive(level)(expr)
```

## Аргументы

- **expr** — выражение, зависящее от значений столбцов. Возвращает данные **числовых типов**, **Date** или **DateTime**.

## Параметры

- **level** — уровень квантиля. Необязательный параметр. Возможные значения: (0, 1) — граничные значения не учитываются. Значение по умолчанию: 0.5. При **level=0.5** функция вычисляет **медиану**. **Float**.

## Возвращаемое значение

- Квантиль заданного уровня.

Тип:

- **Float64** для входных данных числового типа.
- **Date**, если входные значения имеют тип **Date**.
- **DateTime**, если входные значения имеют тип **DateTime**.

## Пример

Запрос:

```
CREATE TABLE num AS numbers(1000);
SELECT quantileExactExclusive(0.6)(x) FROM (SELECT number AS x FROM num);
```

Результат:

```
quantileExactExclusive(0.6)(x)
599.6 |
```

## quantileExactInclusive

Точно вычисляет **квантиль** числовой последовательности.

Чтобы получить точный результат, все переданные значения собираются в массив, который затем частично сортируется. Таким образом, функция потребляет объем памяти  $O(n)$ , где  $n$  — количество переданных значений. Для небольшого числа значений эта функция эффективна.

Эта функция эквивалентна Excel функции **PERCENTILE.INC**, тип **R7**.

Если в одном запросе вызывается несколько функций `quantileExactInclusive` с разными значениями `level`, эти функции вычисляются независимо друг от друга. В таких случаях используйте функцию `quantilesExactInclusive`, запрос будет выполняться эффективнее.

### Синтаксис

```
quantileExactInclusive(level)(expr)
```

### Аргументы

- `expr` — выражение, зависящее от значений столбцов. Возвращает данные **числовых типов**, **Date** или **DateTime**.

### Параметры

- `level` — уровень квантиля. Необязательный параметр. Возможные значения:  $[0, 1]$  — граничные значения учитываются. Значение по умолчанию: 0.5. При `level=0.5` функция вычисляет **медиану**. **Float**.

### Возвращаемое значение

- Квантиль заданного уровня.

Тип:

- **Float64** для входных данных числового типа.
- **Date**, если входные значения имеют тип **Date**.
- **DateTime**, если входные значения имеют тип **DateTime**.

## Пример

Запрос:

```
CREATE TABLE num AS numbers(1000);
SELECT quantileExactInclusive(0.6)(x) FROM (SELECT number AS x FROM num);
```

Результат:

```
quantileExactInclusive(0.6)(x)
599.4
```

## Смотрите также

- [median](#)
- [quantiles](#)

## quantileExactWeighted

Точно вычисляет [квантиль](#) числовой последовательности, учитывая вес каждого её элемента.

Чтобы получить точный результат, все переданные значения собираются в массив, который затем частично сортируется. Для каждого значения учитывается его вес (количество значений в выборке). В алгоритме используется хэш-таблица. Таким образом, если переданные значения часто повторяются, функция потребляет меньше оперативной памяти, чем [quantileExact](#). Эту функцию можно использовать вместо [quantileExact](#) если указать вес 1.

Внутренние состояния функций [quantile\\*](#) не объединяются, если они используются в одном запросе. Если вам необходимо вычислить квантили нескольких уровней, используйте функцию [quantiles](#), это повысит эффективность запроса.

### Синтаксис

```
quantileExactWeighted(level)(expr, weight)
```

Алиас: `medianExactWeighted`.

### Аргументы

- `level` — уровень квантили. Опционально. Константное значение с плавающей запятой от 0 до 1. Мы рекомендуем использовать значение `level` из диапазона [0.01, 0.99]. Значение по умолчанию: 0.5. При `level=0.5` функция вычисляет [медиану](#).
- `expr` — выражение, зависящее от значений столбцов, возвращающее данные [числовых типов](#) или типов [Date](#), [DateTime](#).
- `weight` — столбец с весами элементов последовательности. Вес — это количество повторений элемента в последовательности.

### Возвращаемое значение

- Quantile of the specified level.

Тип:

- [Float64](#) для входных данных числового типа.
- [Date](#), если входные значения имеют тип `Date`.

- [DateTime](#), если входные значения имеют тип `DateTime`.

## Пример

Входная таблица:

n	val
0	3
1	2
2	1
5	4

Запрос:

```
SELECT quantileExactWeighted(n, val) FROM t
```

Результат:

```
quantileExactWeighted(n, val)---  
      1 |
```

## Смотрите также

- [median](#)
- [quantiles](#)

# quantileTiming

Вычисляет [квантиль](#) числовой последовательности с детерминированной точностью.

Результат детерминирован (не зависит от порядка обработки запроса). Функция оптимизирована для работы с последовательностями, описывающими такие распределения, как время загрузки веб-страниц или время отклика бэкенда.

Внутренние состояния функций `quantile*` не объединяются, если они используются в одном запросе. Если вам необходимо вычислить квантили нескольких уровней, используйте функцию [quantiles](#), это повысит эффективность запроса.

## Синтаксис

```
quantileTiming(level)(expr)
```

Алиас: `medianTiming`.

## Аргументы

- `level` — уровень квантили. Опционально. Константное значение с плавающей запятой от 0 до 1. Мы рекомендуем использовать значение `level` из диапазона [0.01, 0.99]. Значение по умолчанию: 0.5. При `level=0.5` функция вычисляет [медиану](#).

- `expr` — выражение, зависящее от значений столбцов, возвращающее данные типа **Float\***.

- Если в функцию передать отрицательные значения, то её поведение не определено.
- Если значение больше, чем 30 000 (например, время загрузки страницы превышает 30 секунд), то оно приравнивается к 30 000.

## Точность

Вычисления точны при соблюдении следующих условий:

- Размер выборки не превышает 5670 элементов.
- Размер выборки превышает 5670 элементов, но значение каждого элемента не больше 1024.

В противном случае, результат вычисления округляется до ближайшего множителя числа 16.

## Примечание

Для указанного типа последовательностей функция производительнее и точнее, чем **quantile**.

## Возвращаемое значение

- Квантиль заданного уровня.

Тип: `Float32`.

## Примечания

Если в функцию `quantileTimingIf` не передать значений, то вернётся **NaN**. Это необходимо для отделения подобных случаев от случаев, когда результат 0. Подробности про сортировку `NaN` смотрите в разделе **Секция ORDER BY**.

## Пример

Входная таблица:

response_time
72
112
126
145
104
242
313
168
108

Запрос:

```
SELECT quantileTiming(response_time) FROM t
```

Результат:

```
quantileTiming(response_time)─  
126 |
```

## Смотрите также

- [median](#)
- [quantiles](#)

# quantileTimingWeighted

С детерминированной точностью вычисляет [квантиль](#) числовой последовательности, учитывая вес каждого элемента.

Результат детерминирован (не зависит от порядка обработки запроса). Функция оптимизирована для работы с последовательностями, описывающими такие распределения, как время загрузки веб-страниц или время отклика бэкенда.

Внутренние состояния функций `quantile*` не объединяются, если они используются в одном запросе. Если вам необходимо вычислить квантили нескольких уровней, используйте функцию [quantiles](#), это повысит эффективность запроса.

## Синтаксис

```
quantileTimingWeighted(level)(expr, weight)
```

Алиас: `medianTimingWeighted`.

## Аргументы

- `level` — уровень квантили. Опционально. Константное значение с плавающей запятой от 0 до 1. Мы рекомендуем использовать значение `level` из диапазона [0.01, 0.99]. Значение по умолчанию: 0.5. При `level=0.5` функция вычисляет [медиану](#).
- `expr` — [выражение](#), зависящее от значений столбцов, возвращающее данные типа [Float\\*](#).

- Если в функцию передать отрицательные значения, то её поведение не определено.  
- Если значение больше, чем 30 000 (например, время загрузки страницы превышает 30 секунд), то оно приравнивается к 30 000.

- `weight` — столбец с весами элементов последовательности. Вес — это количество повторений элемента в последовательности.

## Точность

Вычисления точны при соблюдении следующих условий:

- Размер выборки не превышает 5670 элементов.
- Размер выборки превышает 5670 элементов, но значение каждого элемента не больше 1024.

В противном случае, результат вычисления округляется до ближайшего множителя числа 16.

## Примечание

Для указанного типа последовательностей функция производительнее и точнее, чем [quantile](#).

## Возвращаемое значение

- Квантиль заданного уровня.

Тип: `Float32`.

## Примечания

Если в функцию `quantileTimingIf` не передать значений, то вернётся **`NaN`**. Это необходимо для отделения подобных случаев от случаев, когда результат 0. Подробности про сортировку `NaN` смотрите в разделе **Секция ORDER BY**.

## Пример

Входная таблица:

response_time	weight
68	1
104	2
112	3
126	2
138	1
162	1

Запрос:

```
SELECT quantileTimingWeighted(response_time, weight) FROM t
```

Результат:

```
quantileTimingWeighted(response_time, weight)
112 |
```

## Смотрите также

- [median](#)
- [quantiles](#)

## quantileDeterministic

Приблизительно вычисляет **квантиль** числовой последовательности.

Функция использует алгоритм **reservoir sampling** с размером резервуара до 8192 и детерминированным алгоритмом сэмплирования. Результат детерминирован. Чтобы получить точную квантиль используйте функцию `quantileExact`.

Внутренние состояния функций `quantile*` не объединяются, если они используются в одном запросе. Если вам необходимо вычислить квантили нескольких уровней, используйте функцию `quantiles`, это повысит эффективность запроса.

## Синтаксис

```
quantileDeterministic(level)(expr, determinator)
```

Алиас: `medianDeterministic`.

## Аргументы

- `level` — уровень квантили. Опционально. Константное значение с плавающей запятой от 0 до 1. Мы рекомендуем использовать значение `level` из диапазона [0.01, 0.99]. Значение по умолчанию: 0.5. При `level=0.5` функция вычисляет **медиану**.
- `expr` — выражение, зависящее от значений столбцов, возвращающее данные **числовых типов** или типов `Date`, `DateTime`.
- `determinator` — число, хэш которого используется при сэмплировании в алгоритме «Reservoir sampling», чтобы сделать результат детерминированным. В качестве значения можно использовать любое определённое положительное число, например, идентификатор пользователя или события. Если одно и то же значение попадается в выборке слишком часто, то функция выдаёт некорректный результат.

## Возвращаемое значение

- Приблизительный квантиль заданного уровня.

Тип:

- `Float64` для входных данных числового типа.
- `Date`, если входные значения имеют тип `Date`.
- `DateTime`, если входные значения имеют тип `DateTime`.

## Пример

Входная таблица:

val
1
1
2
3

Запрос:

```
SELECT quantileDeterministic(val, 1) FROM t
```

Результат:

```
quantileDeterministic(val, 1)─  
1.5 |
```

## Смотрите также

- [median](#)
- [quantiles](#)

# quantileTDigest

Приблизительно вычисляет **квантиль** числовой последовательности, используя алгоритм **t-digest**.

Максимальная ошибка 1%. Потребление памяти —  $\log(n)$ , где  $n$  — число значений. Результат не детерминирован и зависит от порядка выполнения запроса.

Производительность функции ниже, чем производительность функции **quantile** или **quantileTiming**. По соотношению размера состояния к точности вычисления, эта функция значительно превосходит **quantile**.

Внутренние состояния функций **quantile\*** не объединяются, если они используются в одном запросе. Если вам необходимо вычислить квантили нескольких уровней, используйте функцию **quantiles**, это повысит эффективность запроса.

## Синтаксис

```
quantileTDigest(level)(expr)
```

Алиас: `medianTDigest`.

## Аргументы

- `level` — уровень квантили. Опционально. Константное значение с плавающей запятой от 0 до 1. Мы рекомендуем использовать значение `level` из диапазона [0.01, 0.99]. Значение по умолчанию: 0.5. При `level=0.5` функция вычисляет **медиану**.
- `expr` — выражение, зависящее от значений столбцов, возвращающее данные **числовых типов** или типов **Date**, **DateTime**.

## Возвращаемое значение

- Приблизительную квантиль заданного уровня.

Тип:

- **Float64** для входных данных числового типа.
- **Date**, если входные значения имеют тип **Date**.
- **DateTime**, если входные значения имеют тип **DateTime**.

## Пример

Запрос:

```
SELECT quantileTDigest(number) FROM numbers(10)
```

Результат:

```
quantileTDigest(number)─  
4.5 |
```

## Смотрите также

- [median](#)
- [quantiles](#)

# quantileTDigestWeighted

Приблизительно вычисляет **квантиль** числовой последовательности, используя алгоритм **t-digest**. Функция учитывает вес каждого элемента последовательности.

Максимальная ошибка 1%. Потребление памяти —  $\log(n)$ , где  $n$  — число значений. Результат не детерминирован и зависит от порядка выполнения запроса.

Производительность функции ниже, чем производительность функции **quantile** или **quantileTiming**. По соотношению размера состояния к точности вычисления, эта функция значительно превосходит **quantile**.

Внутренние состояния функций **quantile\*** не объединяются, если они используются в одном запросе. Если вам необходимо вычислить квантили нескольких уровней, используйте функцию **quantiles**, это повысит эффективность запроса.

## Примечание

Использование **quantileTDigestWeighted** **не рекомендуется для небольших наборов данных** и может привести к значительной ошибке. Рассмотрите возможность использования **quantileTDigest** в таких случаях.

## Синтаксис

```
quantileTDigestWeighted(level)(expr, weight)
```

Синоним: `medianTDigestWeighted`.

## Аргументы

- `level` — уровень квантили. Опционально. Константное значение с плавающей запятой от 0 до 1. Мы рекомендуем использовать значение `level` из диапазона [0.01, 0.99]. Значение по умолчанию: 0.5. При `level=0.5` функция вычисляет **медиану**.
- `expr` — выражение, зависящее от значений столбцов, возвращающее данные **числовых типов** или типов **Date**, **DateTime**.
- `weight` — столбец с весами элементов последовательности. Вес — это количество повторений элемента в последовательности.

## Возвращаемое значение

- Приблизительный квантиль заданного уровня.

Тип:

- **Float64** для входных данных числового типа.
- **Date**, если входные значения имеют тип `Date`.
- **DateTime**, если входные значения имеют тип `DateTime`.

## Пример

Запрос:

```
SELECT quantileTDigestWeighted(number, 1) FROM numbers(10)
```

Результат:

```
quantileTDigestWeighted(number, 1)─  
4.5 |
```

## Смотрите также

- [median](#)
- [quantiles](#)

## quantileBFloat16

Приближенно вычисляет **квантиль** выборки чисел в формате **bfloat16**. **bfloat16** — это формат с плавающей точкой, в котором для представления числа используется 1 знаковый бит, 8 бит для порядка и 7 бит для мантиссы.

Функция преобразует входное число в 32-битное с плавающей точкой и обрабатывает его старшие 16 бит. Она вычисляет квантиль в формате **bfloat16** и преобразует его в 64-битное число с плавающей точкой, добавляя нулевые биты.

Эта функция выполняет быстрые приближенные вычисления с относительной ошибкой не более 0.390625%.

### Синтаксис

```
quantileBFloat16[(level)](#sql-reference-aggregate-functions-reference-expr)
```

Синоним: [medianBFloat16](#)

### Аргументы

- **expr** — столбец с числовыми данными. [Integer](#), [Float](#).

### Параметры

- **level** — уровень квантиля. Необязательный параметр. Допустимый диапазон значений от 0 до 1. Значение по умолчанию: 0.5. [Float](#).

### Возвращаемое значение

- Приближенное значение квантиля.

Тип: [Float64](#).

### Пример

В таблице есть столбцы с целыми числами и с числами с плавающей точкой:

a	b
1	1.001
2	1.002
3	1.003
4	1.004

Запрос для вычисления 0.75-квантиля (верхнего квартиля):

```
SELECT quantileBFloat16(0.75)(a), quantileBFloat16(0.75)(b) FROM example_table;
```

Результат:

```
quantileBFloat16(0.75)(a) quantileBFloat16(0.75)(b)
 3 |           1 |
```

Обратите внимание, что все числа с плавающей точкой в примере были округлены до 1.0 при преобразовании к `bfloat16`.

## quantileBFloat16Weighted

Версия функции `quantileBFloat16`, которая учитывает вес каждого элемента последовательности.

### См. также

- [median](#)
- [quantiles](#)

## simpleLinearRegression

Выполняет простую (одномерную) линейную регрессию.

```
simpleLinearRegression(x, y)
```

Параметры:

- `x` — столбец со значениями зависимой переменной.
- `y` — столбец со значениями наблюдаемой переменной.

Возвращаемые значения:

Константы (`a`, `b`) результирующей прямой  $y = a*x + b$ .

### Примеры

```
SELECT arrayReduce('simpleLinearRegression', [0, 1, 2, 3], [0, 1, 2, 3])
```

```
arrayReduce('simpleLinearRegression', [0, 1, 2, 3], [0, 1, 2, 3])
          |
```

```
SELECT arrayReduce('simpleLinearRegression', [0, 1, 2, 3], [3, 4, 5, 6])
```

```
arrayReduce('simpleLinearRegression', [0, 1, 2, 3], [3, 4, 5, 6])
          |
```

# stochasticLinearRegression

Функция реализует стохастическую линейную регрессию. Поддерживает пользовательские параметры для скорости обучения, коэффициента регуляризации L2, размера mini-batch и имеет несколько методов обновления весов ([Adam](#) (по умолчанию), [simple SGD](#), [Momentum](#), [Nesterov](#)).

## Параметры

Есть 4 настраиваемых параметра. Они передаются в функцию последовательно, однако не обязательно указывать все, используются значения по умолчанию, однако хорошая модель требует некоторой настройки параметров.

```
stochasticLinearRegression(1.0, 1.0, 10, 'SGD')
```

- Скорость обучения — коэффициент длины шага, при выполнении градиентного спуска. Слишком большая скорость обучения может привести к бесконечным весам модели. По умолчанию 0.00001.
- Коэффициент регуляризации l2. Помогает предотвратить подгонку. По умолчанию 0.1.
- Размер mini-batch задаёт количество элементов, чьи градиенты будут вычислены и просуммированы при выполнении одного шага градиентного спуска. Чистый стохастический спуск использует один элемент, однако использование mini-batch (около 10 элементов) делает градиентные шаги более стабильными. По умолчанию 15.
- Метод обновления весов, можно выбрать один из следующих: [Adam](#) (по умолчанию), [SGD](#), [Momentum](#), [Nesterov](#). [Momentum](#) и [Nesterov](#) более требовательные к вычислительным ресурсам и памяти, однако они имеют высокую скорость сходимости и устойчивости методов стохастического градиента.

## Использование

`stochasticLinearRegression` используется на двух этапах: построение модели и предсказание новых данных. Чтобы построить модель и сохранить её состояние для дальнейшего использования, мы используем комбинатор `-State`.

Для прогнозирования мы используем функцию `evalMLMethod`, которая принимает в качестве аргументов состояние и свойства для прогнозирования.

### 1. Построение модели

Пример запроса:

```
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;
```

Здесь нам также нужно вставить данные в таблицу `train_data`. Количество параметров не фиксировано, оно зависит только от количества аргументов, переданных в `linearRegressionState`. Все они должны быть числовыми значениями.

Обратите внимание, что столбец с целевым значением (которое мы хотели бы научиться предсказывать) вставляется в качестве первого аргумента.

### 2. Прогнозирование

После сохранения состояния в таблице мы можем использовать его несколько раз для прогнозирования или смёржить с другими состояниями и создать новые, улучшенные модели.

```
WITH (SELECT state FROM your_model) AS model SELECT  
evalMLMethod(model, param1, param2) FROM test_data
```

Запрос возвращает столбец прогнозируемых значений. Обратите внимание, что первый аргумент `evalMLMethod` это объект `AggregateFunctionState`, далее идут столбцы свойств.

`test_data` — это таблица, подобная `train_data`, но при этом может не содержать целевое значение.

## Примечания

- Объединить две модели можно следующим запросом:

```
SELECT state1 + state2 FROM your_models
```

где таблица `your_models` содержит обе модели. Запрос вернёт новый объект `AggregateFunctionState`.

- Пользователь может получать веса созданной модели для своих целей без сохранения модели, если не использовать комбинатор `-State`.

```
SELECT stochasticLinearRegression(0.01)(target, param1, param2) FROM train_data
```

Подобный запрос строит модель и возвращает её веса, отвечающие параметрам моделей и смещение. Таким образом, в приведенном выше примере запрос вернет столбец с тремя значениями.

## Смотрите также

- [stochasticLogisticRegression](#)
- [Отличие линейной от логистической регрессии.](#)

## stochasticLogisticRegression

Функция реализует стохастическую логистическую регрессию. Её можно использовать для задачи бинарной классификации, функция поддерживает те же пользовательские параметры, что и `stochasticLinearRegression` и работает таким же образом.

## Параметры

Параметры те же, что и в `stochasticLinearRegression`:

`learning rate`, `l2 regularization coefficient`, `mini-batch size`, `method for updating weights`.

Смотрите раздел [parameters](#).

```
stochasticLogisticRegression(1.0, 1.0, 10, 'SGD')
```

- Построение модели

Смотрите раздел [Построение модели](#) в описании [stochasticLinearRegression](#).

Прогнозируемые метки должны быть в диапазоне `\[-1, 1\]`.

## 1. Прогнозирование

Используя сохраненное состояние, можно предсказать вероятность наличия у объекта метки 1.

```
```sql
WITH (SELECT state FROM your_model) AS model SELECT
evalMLMethod(model, param1, param2) FROM test_data
```
```

Запрос возвращает столбец вероятностей. Обратите внимание, что первый аргумент `evalMLMethod` это объект `AggregateFunctionState`, далее идут столбцы свойств.

Мы также можем установить границу вероятности, которая присваивает элементам различные метки.

```
```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)
```
```

Тогда результатом будут метки.

`test_data` — это таблица, подобная `train_data`, но при этом может не содержать целевое значение.

### Смотрите также

- [stochasticLinearRegression](#)
- [Отличие линейной от логистической регрессии](#)

## 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

Вычисляет t-критерий Стьюдента для выборок из двух генеральных совокупностей.

### Синтаксис

```
studentTTest(sample_data, sample_index)
```

Значения выборок берутся из столбца `sample_data`. Если `sample_index` равно 0, то значение из этой строки принадлежит первой выборке. Во всех остальных случаях значение принадлежит второй выборке.

Проверяется нулевая гипотеза, что средние значения генеральных совокупностей совпадают. Для применения t-критерия Стьюдента распределение в генеральных совокупностях должно быть нормальным и дисперсии должны совпадать.

### Аргументы

- `sample_data` — данные выборок. **Integer**, **Float** or **Decimal**.
- `sample_index` — индексы выборок. **Integer**.

## Возвращаемые значения

Кортеж с двумя элементами:

- вычисленное значение критерия Стьюдента. **Float64**.
- вычисленное р-значение. **Float64**.

## Пример

Таблица:

| sample_data | sample_index |
|-------------|--------------|
| 20.3        | 0            |
| 21.1        | 0            |
| 21.9        | 1            |
| 21.7        | 0            |
| 19.9        | 1            |
| 21.8        | 1            |

Запрос:

```
SELECT studentTTest(sample_data, sample_index) FROM student_ttest;
```

Результат:

```
studentTTest(sample_data, sample_index)
(-0.21739130434783777,0.8385421208415731) |
```

## Смотрите также

- [t-критерий Стьюдента](#)
- [welchTTest](#)

# welchTTest

Вычисляет t-критерий Уэлча для выборок из двух генеральных совокупностей.

## Синтаксис

```
welchTTest(sample_data, sample_index)
```

Значения выборок берутся из столбца `sample_data`. Если `sample_index` равно 0, то значение из этой строки принадлежит первой выборке. Во всех остальных случаях значение принадлежит второй выборке.

Проверяется нулевая гипотеза, что средние значения генеральных совокупностей совпадают. Для применения t-критерия Уэлча распределение в генеральных совокупностях должно быть нормальным. Дисперсии могут не совпадать.

## Аргументы

- `sample_data` — данные выборок. [Integer](#), [Float](#) or [Decimal](#).
- `sample_index` — индексы выборок. [Integer](#).

## Возвращаемые значения

**Кортеж** с двумя элементами:

- вычисленное значение критерия Уэлча. [Float64](#).
- вычисленное р-значение. [Float64](#).

## Пример

Таблица:

| sample_data | sample_index |
|-------------|--------------|
| 20.3        | 0            |
| 22.1        | 0            |
| 21.9        | 0            |
| 18.9        | 1            |
| 20.3        | 1            |
| 19          | 1            |

Запрос:

```
SELECT welchTTest(sample_data, sample_index) FROM welch_ttest;
```

Результат:

```
welchTTest(sample_data, sample_index)
(2.7988719532211235, 0.051807360348581945)
```

## Смотрите также

- [t-критерий Уэлча](#)
- [studentTTest](#)

# entropy

Вычисляет [информационную энтропию](#) столбца данных.

## Синтаксис

```
entropy(val)
```

## Аргументы

- `val` — столбец значений любого типа

## Возвращаемое значение

- Информационная энтропия.

Тип: [Float64](#).

## Пример

Запрос:

```
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;
```

Результат:

| entropy(vals) | entropy(strings) |
|---------------|------------------|
| 1             | 1.75             |

## mannWhitneyUTest

Вычисляет U-критерий Манна — Уитни для выборок из двух генеральных совокупностей.

### Синтаксис

```
mannWhitneyUTest[(alternative[, continuity_correction])](sample_data, sample_index)
```

Значения выборок берутся из столбца `sample_data`. Если `sample_index` равно 0, то значение из этой строки принадлежит первой выборке. Во всех остальных случаях значение принадлежит второй выборке.

Проверяется нулевая гипотеза, что генеральные совокупности стохастически равны. Наряду с двусторонней гипотезой могут быть проверены и односторонние.

Для применения U-критерия Манна — Уитни закон распределения генеральных совокупностей не обязан быть нормальным.

### Аргументы

- `sample_data` — данные выборок. **Integer**, **Float** или **Decimal**.
- `sample_index` — индексы выборок. **Integer**.

### Параметры

- `alternative` — альтернативная гипотеза. (Необязательный параметр, по умолчанию: `'two-sided'`.) **String**.
  - `'two-sided'`;
  - `'greater'`;
  - `'less'`.
- `continuity_correction` — если не 0, то при вычислении р-значения применяется коррекция непрерывности. (Необязательный параметр, по умолчанию: 1.) **UInt64**.

### Возвращаемые значения

**Кортеж** с двумя элементами:

- вычисленное значение критерия Манна — Уитни. **Float64**.
- вычисленное р-значение. **Float64**.

## Пример

Таблица:

| sample_data | sample_index |
|-------------|--------------|
| 10          | 0            |
| 11          | 0            |
| 12          | 0            |
| 1           | 1            |
| 2           | 1            |
| 3           | 1            |

Запрос:

```
SELECT mannWhitneyUTest('greater')(sample_data, sample_index) FROM mww_ttest;
```

Результат:

```
mannWhitneyUTest('greater')(sample_data, sample_index)
(9,0.04042779918503192)
```

## Смотрите также

- [U-критерий Манна — Уитни](#)

# median

Функции `median*` — синонимы для соответствующих функций `quantile*`. Они вычисляют медиану числовой последовательности.

Функции:

- `median` — синоним для [quantile](#).
- `medianDeterministic` — синоним для [quantileDeterministic](#).
- `medianExact` — синоним для [quantileExact](#).
- `medianExactWeighted` — синоним для [quantileExactWeighted](#).
- `medianTiming` — синоним для [quantileTiming](#).
- `medianTimingWeighted` — синоним для [quantileTimingWeighted](#).
- `medianTDigest` — синоним для [quantileTDigest](#).
- `medianTDigestWeighted` — синоним для [quantileTDigestWeighted](#).
- `medianBFloat16` — синоним для [quantileBFloat16](#).

## Пример

Входная таблица:

| val |
|-----|
| 1   |
| 1   |
| 2   |
| 3   |

Запрос:

```
SELECT medianDeterministic(val, 1) FROM t;
```

Результат:

```
medianDeterministic(val, 1)─  
1.5 |
```

## Комбинаторы агрегатных функций

К имени агрегатной функции может быть приписан некоторый суффикс. При этом, работа агрегатной функции некоторым образом модифицируется.

### -If

К имени любой агрегатной функции может быть приписан суффикс -If. В этом случае, агрегатная функция принимает ещё один дополнительный аргумент - условие (типа UInt8). Агрегатная функция будет обрабатывать только те строки, для которых условие сработало. Если условие ни разу не сработало - возвращается некоторое значение по умолчанию (обычно - нули, пустые строки).

Примеры: `sumIf(column, cond)`, `countIf(cond)`, `avgIf(x, cond)`, `quantilesTimingIf(level1, level2)(x, cond)`, `argMinIf(arg, val, cond)` и т. п.

С помощью условных агрегатных функций, вы можете вычислить агрегаты сразу для нескольких условий, не используя подзапросы и JOIN-ы.

Например, в Яндекс.Метрике, условные агрегатные функции используются для реализации функциональности сравнения сегментов.

### -Array

К имени любой агрегатной функции может быть приписан суффикс -Array. В этом случае, агрегатная функция вместо аргументов типов T принимает аргументы типов Array(T) (массивы). Если агрегатная функция принимает несколько аргументов, то это должны быть массивы одинаковых длин. При обработке массивов, агрегатная функция работает, как исходная агрегатная функция по всем элементам массивов.

Пример 1: `sumArray(arr)` - просуммировать все элементы всех массивов arr. В данном примере можно было бы написать проще: `sum(arraySum(arr))`.

Пример 2: `uniqArray(arr)` - посчитать количество уникальных элементов всех массивов arr. Это можно было бы сделать проще: `uniq(arrayJoin(arr))`, но не всегда есть возможность добавить arrayJoin в запрос.

Комбинаторы -If и -Array можно сочетать. При этом, должен сначала идти Array, а потом If.

Примеры: `uniqArrayIf(arr, cond)`, `quantilesTimingArrayIf(level1, level2)(arr, cond)`. Из-за такого порядка получается, что аргумент cond не должен быть массивом.

## -SimpleState

При использовании этого комбинатора агрегатная функция возвращает то же значение, но типа `SimpleAggregateFunction(...)`. Текущее значение функции может храниться в таблице для последующей работы с таблицами семейства `AggregatingMergeTree`.

### Синтаксис

```
<aggFunction>SimpleState(x)
```

### Аргументы

- `x` — параметры агрегатной функции.

### Возвращаемое значение

Значение агрегатной функции типа `SimpleAggregateFunction(...)`.

### Пример

Запрос:

```
WITH anySimpleState(number) AS c SELECT toTypeName(c), c FROM numbers(1);
```

Результат:

```
toTypeName(c)
SimpleAggregateFunction(any, UInt64) | 0 | c
```

## -State

В случае применения этого комбинатора, агрегатная функция возвращает не готовое значение (например, в случае функции `uniq` — количество уникальных значений), а промежуточное состояние агрегации (например, в случае функции `uniq` — хэш-таблицу для расчёта количества уникальных значений), которое имеет тип `AggregateFunction(...)` и может использоваться для дальнейшей обработки или может быть сохранено в таблицу для последующей доагрегации.

Для работы с промежуточными состояниями предназначены:

- Движок таблиц `AggregatingMergeTree`.
- Функция `finalizeAggregation`.
- Функция `runningAccumulate`.
- Комбинатор `-Merge`.
- Комбинатор `-MergeState`.

## -Merge

В случае применения этого комбинатора, агрегатная функция будет принимать в качестве аргумента промежуточное состояние агрегации, доаггрегировать (объединять вместе) эти состояния, и возвращать готовое значение.

## -MergeState

Выполняет слияние промежуточных состояний агрегации, аналогично комбинатору -Merge, но возвращает не готовое значение, а промежуточное состояние агрегации, аналогично комбинатору -State.

## -ForEach

Преобразует агрегатную функцию для таблиц в агрегатную функцию для массивов, которая применяет агрегирование для соответствующих элементов массивов и возвращает массив результатов. Например, `sumForEach` для массивов [1, 2], [3, 4, 5] и [6, 7] даст результат [10, 13, 5], сложив соответственные элементы массивов.

## -Distinct

При наличии комбинатора Distinct, каждое уникальное значение аргументов, будет учтено в агрегатной функции только один раз.

Примеры: `sum(DISTINCT x)`, `groupArray(DISTINCT x)`, `corrStableDistinct(DISTINCT x, y)` и т.п.

## -OrDefault

Изменяет поведение агрегатной функции.

Если на вход агрегатной функции передан пустой набор данных, то с помощью комбинатора -OrDefault функция возвращает значение по умолчанию для соответствующего типа данных. Комбинатор применяется к агрегатным функциям, которые могут принимать пустые входные данные.

-OrDefault можно использовать с другими комбинаторами.

### Синтаксис

```
<aggFunction>OrDefault(x)
```

### Аргументы

- `x` — аргументы агрегатной функции.

### Возвращаемые значения

Возвращает значение по умолчанию для соответствующего типа агрегатной функции, если агрегировать нечего.

Тип данных зависит от используемой агрегатной функции.

### Пример

Запрос:

```
SELECT avg(number), avgOrDefault(number) FROM numbers(0)
```

Результат:

| avg(number) | avgOrDefault(number) |
|-------------|----------------------|
| nan         | 0                    |

Также -OrDefault может использоваться с другими комбинаторами. Это полезно, когда агрегатная функция не принимает пустые входные данные.

Запрос:

```
SELECT avgOrDefaultIf(x, x > 10)
FROM
(
    SELECT toDecimal32(1.23, 2) AS x
)
```

Результат:

```
avgOrDefaultIf(x, greater(x, 10))—
0.00 |
```

## -OrNull

Изменяет поведение агрегатной функции.

Комбинатор преобразует результат агрегатной функции к типу **Nullable**. Если агрегатная функция не получает данных на вход, то с комбинатором она возвращает **NULL**.

`-OrNull` может использоваться с другими комбинаторами.

### Синтаксис

```
<aggFunction>OrNull(x)
```

### Аргументы

- `x` — аргументы агрегатной функции.

### Возвращаемые значения

- Результат агрегатной функции, преобразованный в тип данных **Nullable**.
- **NULL**, если у агрегатной функции нет входных данных.

Тип: `Nullable(aggregate function return type)`.

### Пример

Добавьте `-orNull` в конец агрегатной функции.

Запрос:

```
SELECT sumOrNull(number), toTypeName(sumOrNull(number)) FROM numbers(10) WHERE number > 10
```

Результат:

```
sumOrNull(number)—toTypeName(sumOrNull(number))—
NULL | Nullable(UInt64) |
```

Также `-OrNone` может использоваться с другими комбинаторами. Это полезно, когда агрегатная функция не принимает пустые входные данные.

Запрос:

```
SELECT avgOrNullIf(x, x > 10)
FROM
(
    SELECT toDecimal32(1.23, 2) AS x
)
```

Результат:

```
avgOrNullIf(x, greater(x, 10))
```

NULL |

## -Resample

Позволяет поделить данные на группы, а затем по-отдельности агрегирует данные для этих групп. Группы образуются разбиением значений одного из столбцов на интервалы.

```
<aggFunction>Resample(start, end, step)(<aggFunction_params>, resampling_key)
```

### Аргументы

- `start` — начальное значение для интервала значений `resampling_key`.
- `stop` — конечное значение для интервала значений `resampling_key`. Интервал не включает значение `stop` (`[start, stop)`).
- `step` — шаг деления полного интервала на подинтервалы. Функция `aggFunction` выполняется для каждого из подинтервалов независимо.
- `resampling_key` — столбец, значения которого используются для разделения данных на интервалы.
- `aggFunction_params` — параметры `aggFunction`.

### Возвращаемые значения

- Массив результатов `aggFunction` для каждого подинтервала.

### Пример

Рассмотрим таблицу `people` со следующими данными:

| name   | age | wage |
|--------|-----|------|
| John   | 16  | 10   |
| Alice  | 30  | 15   |
| Mary   | 35  | 8    |
| Evelyn | 48  | 11.5 |
| David  | 62  | 9.9  |
| Brian  | 60  | 16   |

Получим имена людей, чей возраст находится в интервалах [30,60) и [60,75). Поскольку мы используем целочисленное представление возраста, то интервалы будут выглядеть как [30, 59] и [60,74].

Чтобы собрать имена в массив, возьмём агрегатную функцию `groupArray`. Она принимает один аргумент. В нашем случае, это столбец `name`. Функция `groupArrayResample` должна использовать столбец `age` для агрегирования имён по возрасту. Чтобы определить необходимые интервалы, передадим в функцию `groupArrayResample` аргументы `30, 75, 30`.

```
SELECT groupArrayResample(30, 75, 30)(name, age) from people
```

```
groupArrayResample(30, 75, 30)(name, age)———  
[['Alice','Mary','Evelyn'],['David','Brian']] |
```

Посмотрим на результаты.

John не попал в выдачу, поскольку слишком молод. Остальные распределены согласно заданным возрастным интервалам.

Теперь посчитаем общее количество людей и их среднюю заработную плату в заданных возрастных интервалах.

```
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] |
```

## Параметрические агрегатные функции

Некоторые агрегатные функции могут принимать не только столбцы-аргументы (по которым производится свёртка), но и набор параметров - констант для инициализации. Синтаксис - две пары круглых скобок вместо одной. Первая - для параметров, вторая - для аргументов.

### histogram

Рассчитывает адаптивную гистограмму. Не гарантирует точного результата.

```
histogram(number_of_bins)(values)
```

Функция использует [A Streaming Parallel Decision Tree Algorithm](#). Границы столбцов устанавливаются по мере поступления новых данных в функцию. В общем случае столбцы имеют разную ширину.

#### Аргументы

values — выражение, предоставляющее входные значения.

#### Параметры

number\_of\_bins — максимальное количество корзин в гистограмме. Функция автоматически вычисляет количество корзин. Она пытается получить указанное количество корзин, но если не получилось, то в результате корзин будет меньше.

#### Возвращаемые значения

- Массив кортежей следующего вида:

```
...  
[(lower_1, upper_1, height_1), ... (lower_N, upper_N, height_N)]
```

- `lower` — нижняя граница корзины.
- `upper` — верхняя граница корзины.
- `height` — количество значений в корзине.

## Пример

```
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)] |
```

С помощью функции **bar** можно визуализировать гистограмму, например:

```
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  
)
```



В этом случае необходимо помнить, что границы корзин гистограммы не известны.

## sequenceMatch(pattern)(timestamp, cond1, cond2, ...)

Проверяет, содержит ли последовательность событий цепочку, которая соответствует указанному шаблону.

```
sequenceMatch(pattern)(timestamp, cond1, cond2, ...)
```

## Предупреждение

События, произошедшие в одну и ту же секунду, располагаются в последовательности в неопределенном порядке, что может повлиять на результат работы функции.

## Аргументы

- `timestamp` — столбец, содержащий метки времени. Типичный тип данных столбца — `Date` или `DateTime`. Также можно использовать любой из поддержанных типов данных `UInt`.
- `cond1, cond2` — условия, описывающие цепочку событий. Тип данных — `UInt8`. Можно использовать до 32 условий. Функция учитывает только те события, которые указаны в условиях. Функция пропускает данные из последовательности, если они не описаны ни в одном из условий.

## Параметры

- `pattern` — строка с шаблоном. Смотрите [Синтаксис шаблонов](#).

## Возвращаемые значения

- 1, если цепочка событий, соответствующая шаблону найдена.
- 0, если цепочка событий, соответствующая шаблону не найдена.

Тип: `UInt8`.

## Синтаксис шаблонов

- `(?N)` — соответствует условию на позиции `N`. Условия пронумерованы по порядку в диапазоне [1, 32]. Например, `(?1)` соответствует условию, заданному параметром `cond1`.
- `.*` — соответствует любому количеству событий. Для этого элемента шаблона не надо задавать условия.
- `(?t operator value)` — устанавливает время в секундах, которое должно разделять два события. Например, шаблон `(?1)(?t>1800)(?2)` соответствует событиям, которые произошли более чем через 1800 секунд друг от друга. Между этими событиями может находиться произвольное количество любых событий. Операторы могут быть `>=`, `>`, `<`, `<=`, `==`.

## Примеры

Пусть таблица `t` содержит следующие данные:

| time | number |
|------|--------|
| 1    | 1      |
| 2    | 3      |
| 3    | 2      |

Выполним запрос:

```
SELECT sequenceMatch('(?1)(?2)')(time, number = 1, number = 2) FROM t
```

```
sequenceMatch('(?1)(?2)')(time, equals(number, 1), equals(number, 2))—  
1 |
```

Функция нашла цепочку событий, в которой число 2 следует за числом 1. Число 3 между ними было пропущено, поскольку оно не было использовано ни в одном из условий.

```
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 |
```

В этом случае функция не может найти цепочку событий, соответствующую шаблону, поскольку событие для числа 3 произошло между 1 и 2. Если бы в этом же случае мы бы проверяли условие на событие для числа 4, то цепочка бы соответствовала шаблону.

```
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 |
```

## Смотрите также

- [sequenceCount](#)

## sequenceCount(pattern)(time, cond1, cond2, ...)

Вычисляет количество цепочек событий, соответствующих шаблону. Функция обнаруживает только непересекающиеся цепочки событий. Она начинает искать следующую цепочку только после того, как полностью совпала текущая цепочка событий.

## Предупреждение

События, произошедшие в одну и ту же секунду, располагаются в последовательности в неопределенном порядке, что может повлиять на результат работы функции.

```
sequenceCount(pattern)(timestamp, cond1, cond2, ...)
```

## Аргументы

- `timestamp` — столбец, содержащий метки времени. Типичный тип данных столбца — `Date` или `DateTime`. Также можно использовать любой из поддержанных типов данных `UInt`.
- `cond1, cond2` — условия, описывающие цепочку событий. Тип данных — `UInt8`. Можно использовать до 32 условий. Функция учитывает только те события, которые указаны в условиях. Функция пропускает данные из последовательности, если они не описаны ни в одном из условий.

## Параметры

- `pattern` — строка с шаблоном. Смотрите [Синтаксис шаблонов](#).

## Возвращаемое значение

- Число непересекающихся цепочек событий, соответствующих шаблону.

Тип: `UInt64`.

## Пример

Пусть таблица `t` содержит следующие данные:

| time | number |
|------|--------|
| 1    | 1      |
| 2    | 3      |
| 3    | 2      |
| 4    | 1      |
| 5    | 3      |
| 6    | 2      |

Вычислим сколько раз число 2 стоит после числа 1, причем между 1 и 2 могут быть любые числа:

```
SELECT sequenceCount('(?1).*(?2)')(time, number = 1, number = 2) FROM t
```

```
sequenceCount('(?1).*(?2)')(time, equals(number, 1), equals(number, 2))—  
2 |
```

## Смотрите также

- [sequenceMatch](#)

## windowFunnel

Отыскивает цепочки событий в скользящем окне по времени и вычисляет максимальное количество произошедших событий из цепочки.

Функция работает по алгоритму:

- Функция отыскивает данные, на которых срабатывает первое условие из цепочки, и присваивает счетчику событий значение 1. С этого же момента начинается отсчет времени скользящего окна.
- Если в пределах окна последовательно попадаются события из цепочки, то счетчик увеличивается. Если последовательность событий нарушается, то счетчик не растет.
- Если в данных оказалось несколько цепочек разной степени завершенности, то функция выдаст только размер самой длинной цепочки.

## Синтаксис

```
windowFunnel(window, [mode, [mode, ... ]])(timestamp, cond1, cond2, ..., condN)
```

## Аргументы

- `timestamp` — имя столбца, содержащего временные отметки. [Date](#), [DateTime](#) и другие параметры с типом [Integer](#). В случае хранения меток времени в столбцах с типом [UInt64](#), максимально допустимое значение соответствует ограничению для типа [Int64](#), т.е. равно  $2^{63}-1$ .
- `cond` — условия или данные, описывающие цепочку событий. [UInt8](#).

## Параметры

- `window` — ширина скользящего окна по времени. Это время между первым и последним условием. Единица измерения зависит от `timestamp` и может варьироваться. Должно соблюдаться условие `timestamp` события `cond1 <= timestamp` события `cond2 <= ... <= timestamp` события `condN <= timestamp` события `cond1 + window`.

- `mode` — необязательный параметр. Может быть установлено несколько значений одновременно.
  - `'strict'` — не учитывать подряд идущие повторяющиеся события.
  - `'strict_order'` — запрещает посторонние события в искомой последовательности. Например, при поиске цепочки A->B->C в A->B->D->C поиск будет остановлен на D и функция вернет 2.
  - `'strict_increase'` — условия применяются только для событий со строго возрастающими временными метками.

## Возвращаемое значение

Максимальное количество последовательно сработавших условий из цепочки в пределах скользящего окна по времени. Исследуются все цепочки в выборке.

Тип: `Integer`.

## Пример

Определим, успевает ли пользователь за установленный период выбрать телефон в интернет-магазине, купить его и сделать повторный заказ.

Зададим следующую цепочку событий:

1. Пользователь вошел в личный кабинет (`eventID = 1001`).
2. Пользователь ищет телефон (`eventID = 1003, product = 'phone'`).
3. Пользователь сделал заказ (`eventID = 1009`)
4. Пользователь сделал повторный заказ (`eventID = 1010`).

Входная таблица:

| 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   |

Сделаем запрос и узнаем, как далеко пользователь `user_id` смог пройти по цепочке за период в январе-феврале 2019-го года.

Запрос:

```

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;

```

# retention

Аналитическая функция, которая показывает, насколько выдерживаются те или иные условия, например, удержание динамики/уровня посещаемости сайта.

Функция принимает набор (от 1 до 32) логических условий, как в **WHERE**, и применяет их к заданному набору данных.

Условия, кроме первого, применяются попарно: результат второго будет истинным, если истинно первое и второе, третьего - если истинно первое и третье и т.д.

## Синтаксис

```
retention(cond1, cond2, ..., cond32)
```

## Аргументы

- `cond` — вычисляемое условие или выражение, которое возвращает `UInt8` результат (1/0).

## Возвращаемое значение

Массив из 1 или 0.

- 1 — условие выполнено.
- 0 — условие не выполнено.

Тип: `UInt8`.

## Пример

Рассмотрим пример расчета функции `retention` для определения посещаемости сайта.

### 1. Создадим таблицу для иллюстрации примера.

```
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);
```

Входная таблица:

Запрос:

```
SELECT * FROM retention_test
```

Ответ:

| 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. Сгруппируем пользователей по уникальному идентификатору uid с помощью функции retention.

Запрос:

```
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
```

Результат:

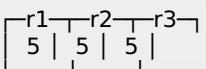
| uid | r       |
|-----|---------|
| 0   | [1,1,1] |
| 1   | [1,1,1] |
| 2   | [1,1,1] |
| 3   | [1,1,1] |
| 4   | [1,1,1] |
| 5   | [0,0,0] |
| 6   | [0,0,0] |
| 7   | [0,0,0] |
| 8   | [0,0,0] |
| 9   | [0,0,0] |
| 10  | [0,0,0] |
| 11  | [0,0,0] |
| 12  | [0,0,0] |
| 13  | [0,0,0] |
| 14  | [0,0,0] |

### 3. Рассчитаем количество посещений сайта за день.

Запрос:

```
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
)
```

Результат:



Где:

- `r1` - количество уникальных посетителей за 2020-01-01 (`cond1`).
- `r2` - количество уникальных посетителей в период между 2020-01-01 и 2020-01-02 (`cond1` и `cond2`).
- `r3` - количество уникальных посетителей в период между 2020-01-01 и 2020-01-03 (`cond1` и `cond3`).

## uniqUpTo(N)(x)

Вычисляет количество различных значений аргумента, если оно меньше или равно N.

В случае, если количество различных значений аргумента больше N, возвращает N + 1.

Рекомендуется использовать для маленьких N - до 10. Максимальное значение N - 100.

Для состояния агрегатной функции используется количество оперативки равное  $1 + N * \text{размер одного значения байт}$ .

Для строк запоминается не криптографический хэш, имеющий размер 8 байт. То есть, для строк вычисление приближённое.

Функция также работает для нескольких аргументов.

Работает максимально быстро за исключением патологических случаев, когда используется большое значение N и количество уникальных значений чуть меньше N.

Пример применения:

Задача: показывать в отчёте только поисковые фразы, по которым было хотя бы 5 уникальных посетителей.  
Решение: пишем в запросе GROUP BY SearchPhrase HAVING uniqUpTo(4)(UserID) >= 5

## sequenceNextNode

Возвращает значение следующего события, соответствующего цепочке событий.

Экспериментальная функция, чтобы включить ее, выполните: `SET allow_experimental_funnel_functions = 1`.

### Синтаксис

```
sequenceNextNode(direction, base)(timestamp, event_column, base_condition, event1, event2, event3, ...)
```

## Параметры

- `direction` — используется для навигации по направлениям.
  - `forward` — двигаться вперед.
  - `backward` — двигаться назад.
- `base` — используется для задания начальной точки.
  - `head` — установить начальную точку на первое событие цепочки.
  - `tail` — установить начальную точку на последнее событие цепочки.
  - `first_match` — установить начальную точку на первое соответствующее событие `event1`.
  - `last_match` — установить начальную точку на последнее соответствующее событие `event1`.

## Аргументы

- `timestamp` — название столбца, содержащего `timestamp`. Поддерживаемые типы данных: `Date`, `DateTime` и другие беззнаковые целые типы.
- `event_column` — название столбца, содержащего значение следующего возвращаемого события. Поддерживаемые типы данных: `String` и `Nullable(String)`.
- `base_condition` — условие, которому должна соответствовать исходная точка.
- `event1, event2, ...` — условия, описывающие цепочку событий. `UInt8`.

## Возвращаемые значения

- `event_column[next_index]` — если есть совпадение с шаблоном и существует следующее значение.
- `NULL` — если нет совпадений с шаблоном или следующего значения не существует.

Тип: `Nullable(String)`.

## Пример

Функцию можно использовать, если есть цепочка событий A->B->C->D->E, и вы хотите определить событие, следующее за B->C, то есть D.

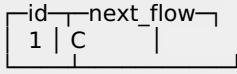
Запрос ищет событие после A->B:

```
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 = 'A', page = 'B') as next_flow FROM
test_flow GROUP BY id;
```

Результат:



## **Поведение для forward и 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   | Comment                                |
|---------------------|----|--------|----------------------------------------|
| 1970-01-01 09:00:01 | 1  | Home   | // Исходная точка, совпадение с Home   |
| 1970-01-01 09:00:02 | 1  | Gift   | // Совпадение с Gift                   |
| 1970-01-01 09:00:03 | 1  | Exit   | // Результат                           |
| 1970-01-01 09:00:01 | 2  | Home   | // Исходная точка, совпадение с Home   |
| 1970-01-01 09:00:02 | 2  | Home   | // Несовпадение с 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   | // Исходная точка, несовпадение с 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 |                                        |

## **Поведение для backward и tail**

```

SELECT id, sequenceNextNode('backward', 'tail')(dt, page, page = 'Basket', page = 'Basket', page = 'Gift') FROM
test_flow GROUP BY id;
    
```

| dt                  | id | page   | Comment                                  |
|---------------------|----|--------|------------------------------------------|
| 1970-01-01 09:00:01 | 1  | Home   |                                          |
| 1970-01-01 09:00:02 | 1  | Gift   |                                          |
| 1970-01-01 09:00:03 | 1  | Exit   | // Исходная точка, несовпадение с Basket |
| 1970-01-01 09:00:01 | 2  | Home   |                                          |
| 1970-01-01 09:00:02 | 2  | Home   | // Результат                             |
| 1970-01-01 09:00:03 | 2  | Gift   | // Совпадение с Gift                     |
| 1970-01-01 09:00:04 | 2  | Basket | // Исходная точка, совпадение с Basket   |
| 1970-01-01 09:00:01 | 3  | Gift   |                                          |
| 1970-01-01 09:00:02 | 3  | Home   | // Результат                             |
| 1970-01-01 09:00:03 | 3  | Gift   | // Исходная точка, совпадение с Gift     |
| 1970-01-01 09:00:04 | 3  | Basket | // Исходная точка, совпадение с Basket   |

## **Поведение для forward и 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 // Исходная точка |
| 1970-01-01 09:00:03 | 1  | Exit // Результат      |

|                     |   |                        |
|---------------------|---|------------------------|
| 1970-01-01 09:00:01 | 2 | Home                   |
| 1970-01-01 09:00:02 | 2 | Home                   |
| 1970-01-01 09:00:03 | 2 | Gift // Исходная точка |
| 1970-01-01 09:00:04 | 2 | Basket Результат       |

|                     |   |                        |
|---------------------|---|------------------------|
| 1970-01-01 09:00:01 | 3 | Gift // Исходная точка |
| 1970-01-01 09:00:02 | 3 | Home // Результат      |
| 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 = '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 // Исходная точка      |
| 1970-01-01 09:00:03 | 1  | Exit // Несовпадение с 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 // Исходная точка        |
| 1970-01-01 09:00:04 | 2 | Basket // Несовпадение с Home |

|                     |   |                           |
|---------------------|---|---------------------------|
| 1970-01-01 09:00:01 | 3 | Gift // Исходная точка    |
| 1970-01-01 09:00:02 | 3 | Home // Совпадение с Home |
| 1970-01-01 09:00:03 | 3 | Gift // Результат         |
| 1970-01-01 09:00:04 | 3 | Basket                    |

## Поведение для backward и 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 // Результат      |
| 1970-01-01 09:00:02 | 1  | Gift // Исходная точка |
| 1970-01-01 09:00:03 | 1  | Exit                   |

|                     |   |                        |
|---------------------|---|------------------------|
| 1970-01-01 09:00:01 | 2 | Home                   |
| 1970-01-01 09:00:02 | 2 | Home // Результат      |
| 1970-01-01 09:00:03 | 2 | Gift // Исходная точка |
| 1970-01-01 09:00:04 | 2 | Basket                 |

|                     |   |                        |
|---------------------|---|------------------------|
| 1970-01-01 09:00:01 | 3 | Gift                   |
| 1970-01-01 09:00:02 | 3 | Home // Результат      |
| 1970-01-01 09:00:03 | 3 | Gift // Исходная точка |
| 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 // Совпадение с Home, результат `Null`
1970-01-01 09:00:02  1  Gift // Исходная точка
1970-01-01 09:00:03  1  Exit

1970-01-01 09:00:01  2  Home // Результат
1970-01-01 09:00:02  2  Home // Совпадение с Home
1970-01-01 09:00:03  2  Gift // Исходная точка
1970-01-01 09:00:04  2  Basket

1970-01-01 09:00:01  3  Gift // Результат
1970-01-01 09:00:02  3  Home // Совпадение с Home
1970-01-01 09:00:03  3  Gift // Исходная точка
1970-01-01 09:00:04  3  Basket

```

## Поведение для 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 // Начало не может быть исходной точкой, поскольку столбец ref не
соответствует '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 // Конец не может быть исходной точкой, поскольку столбец ref не
соответствует '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 // Эта строка не может быть исходной точкой, поскольку столбец ref не
соответствует 'ref3'.
1970-01-01 09:00:02  1  A    ref3 // Исходная точка
1970-01-01 09:00:03  1  B    ref2 // Результат
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 // Результат
1970-01-01 09:00:03 1 B   ref2 // Исходная точка
1970-01-01 09:00:04 1 B   ref1 // Эта строка не может быть исходной точкой, поскольку столбец ref не
соответствует 'ref2'.
```

## Табличные функции

Табличные функции — это метод создания таблиц.

Табличные функции можно использовать в:

- Секции **FROM** запроса **SELECT**.

Это способ создания временной таблицы, которая доступна только в текущем запросе.

- Запросе **CREATE TABLE AS \<table\_function()>**.

Это один из методов создания таблицы.

### Предупреждение

Если настройка **allow\_ddl** выключена, то использовать табличные функции невозможно.

Функция	Описание
file	Создаёт таблицу с движком <b>File</b> .
merge	Создаёт таблицу с движком <b>Merge</b> .
numbers	Создаёт таблицу с единственным столбцом, заполненным целыми числами.
remote	Предоставляет доступ к удалённым серверам, не создавая таблицу с движком <b>Distributed</b> .
url	Создаёт таблицу с движком <b>Url</b> .
mysql	Создаёт таблицу с движком <b>MySQL</b> .
jdbc	Создаёт таблицу с движком <b>JDBC</b> .
odbc	Создаёт таблицу с движком <b>ODBC</b> .
hdfs	Создаёт таблицу с движком <b>HDFS</b> .
s3	Создаёт таблицу с движком <b>S3</b> .

# file

Создаёт таблицу из файла. Данная табличная функция похожа на табличные функции `url` и `hdfs`.

Функция `file` может использоваться в запросах `SELECT` и `INSERT` при работе с движком таблиц `File`.

## Синтаксис

```
file(path, format, structure)
```

## Параметры

- `path` — относительный путь до файла от `user_files_path`. Путь к файлу поддерживает следующие шаблоны в режиме доступа только для чтения `*`, `?`, `{abc,def}` и `{N..M}`, где `N, M` — числа, `'abc'`, `'def'` — строки.
- `format` — **формат** файла.
- `structure` — структура таблицы. Формат: `'column1_name column1_type, column2_name column2_type, ...'`.

## Возвращаемое значение

Таблица с указанной структурой, предназначенная для чтения или записи данных в указанном файле.

## Примеры

Настройка `user_files_path` и содержимое файла `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
```

Получение данных из таблицы в файле `test.csv` и выборка первых двух строк из неё:

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

column1	column2	column3
1	2	3
3	2	1

Получение первых 10 строк таблицы, содержащей 3 столбца типа `UInt32`, из CSV-файла:

```
SELECT * FROM file('test.csv', 'CSV', 'column1 UInt32, column2 UInt32, column3 UInt32') LIMIT 10;
```

Вставка данных из файла в таблицу:

```
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

## Шаблоны поиска в компонентах пути

При описании пути к файлу могут использоваться шаблоны поиска. Обрабатываются только те файлы, у которых путь и название соответствуют шаблону полностью (а не только префикс или суффикс).

- `*` — заменяет любое количество любых символов кроме `/`, включая отсутствие символов.
- `?` — заменяет ровно один любой символ.
- `{some_string,another_string,yet_another_one}` — заменяет любую из строк `'some_string'`, `'another_string'`, `'yet_another_one'`.
- `{N..M}` — заменяет любое число в интервале от `N` до `M` включительно (может содержать ведущие нули).

Конструкция с `{}` аналогична табличной функции `remote`.

### Пример

Предположим, у нас есть несколько файлов со следующими относительными путями:

- `'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'`

Запросим количество строк в этих файлах:

```
SELECT count(*) FROM file('{some,another}_dir/some_file_{1..3}', 'TSV', 'name String, value UInt32');
```

Запросим количество строк во всех файлах этих двух директорий:

```
SELECT count(*) FROM file('{some,another}_dir/*', 'TSV', 'name String, value UInt32');
```

### Предупреждение

Если ваш список файлов содержит интервал с ведущими нулями, используйте конструкцию с фигурными скобками для каждой цифры по отдельности или используйте `?`.

### Пример

Запрос данных из файлов с именами `file000`, `file001`, ..., `file999`:

```
SELECT count(*) FROM file('big_dir/file{0..9}{0..9}{0..9}', 'CSV', 'name String, value UInt32');
```

## Виртуальные столбцы

- `_path` — путь к файлу.
- `_file` — имя файла.

### Смотрите также

- [Виртуальные столбцы](#)

## merge

`merge(db_name, 'tables_regexp')` - создаёт временную таблицу типа Merge. Подробнее смотрите раздел «[Движки таблиц, Merge](#)».

Структура таблицы берётся из первой попавшейся таблицы, подходящей под регулярное выражение.

## numbers

`numbers(N)` - возвращает таблицу с единственным столбцом `number` (`UInt64`), содержащим натуральные числа от 0 до `N-1`.

`numbers(N, M)` - возвращает таблицу с единственным столбцом `number` (`UInt64`), содержащим натуральные числа от `N` до `(N + M - 1)`.

Так же как и таблица `system.numbers` может использоваться для тестов и генерации последовательных значений. Функция `numbers(N, M)` работает более эффективно, чем выборка из `system.numbers`.

Следующие запросы эквивалентны:

```
SELECT * FROM numbers(10);
SELECT * FROM numbers(0,10);
SELECT * FROM system.numbers LIMIT 10;
```

Примеры:

```
-- генерация последовательности всех дат от 2010-01-01 до 2010-12-31
select toDate('2010-01-01') + number as d FROM numbers(365);
```

## remote, remoteSecure

Позволяет обратиться к удалённым серверам без создания таблицы типа [Distributed](#). Функция `remoteSecure` работает аналогично `remote`, но использует защищенное соединение.

Обе функции могут использоваться в запросах `SELECT` и `INSERT`.

### Синтаксис

```
remote('addresses_expr', db, table[, 'user'][, 'password'])  
remote('addresses_expr', db.table[, 'user'][, 'password'])  
remoteSecure('addresses_expr', db, table[, 'user'][, 'password'])  
remoteSecure('addresses_expr', db.table[, 'user'][, 'password'])
```

## Параметры

- `addresses_expr` — выражение, генерирующее адреса удалённых серверов. Это может быть просто один адрес сервера. Адрес сервера — это `host:port` или только `host`.

Вместо параметра `host` может быть указано имя сервера или его адрес в формате IPv4 или IPv6. IPv6 адрес указывается в квадратных скобках.

`port` — TCP-порт удалённого сервера. Если порт не указан, используется `tcp_port` из конфигурационного файла сервера, к которому обратились через функцию `remote` (по умолчанию — 9000), и `tcp_port_secure`, к которому обратились через функцию `remoteSecure` (по умолчанию — 9440).

С IPv6-адресом обязательно нужно указывать порт.

Тип: `String`.

- `db` — имя базы данных. Тип: `String`.
- `table` — имя таблицы. Тип: `String`.
- `user` — имя пользователя. Если пользователь не указан, то по умолчанию `default`. Тип: `String`.
- `password` — пароль. Если пароль не указан, то используется пустой пароль. Тип: `String`.
- `sharding_key` — ключ шардирования для поддержки распределения данных между узлами. Например: `insert into remote('127.0.0.1:9000,127.0.0.2', db, table, 'default', rand())` Тип: `UInt32`.

## Возвращаемое значение

Набор данных с удаленных серверов.

## Использование

Использование табличной функции `remote` менее оптимально, чем создание таблицы типа `Distributed`, так как в этом случае соединения с серверами устанавливаются заново при каждом запросе. Если указываются имена серверов, то приходится также выполнять поиск сервера по имени. Кроме того, не ведётся сквозной подсчёт ошибок при работе с разными репликами. При обработке большого количества запросов всегда создавайте таблицу типа `Distributed`, использовать табличную функцию `remote` в таких случаях не рекомендуется.

Табличная функция `remote` может быть полезна в следующих случаях:

- Обращение на конкретный сервер для сравнения данных, отладки и тестирования.
- Запросы между разными кластерами ClickHouse для исследований.
- Нечастые распределённые запросы, задаваемые вручную.
- Распределённые запросы, где набор серверов определяется каждый раз заново.

## Адреса

```
example01-01-1
example01-01-1:9000
localhost
127.0.0.1
[::]:9000
[2a02:6b8:0:1111::11]:9000
```

Адреса можно указать через запятую. В этом случае ClickHouse обработает запрос как распределённый, т.е. отправит его по всем указанным адресам как на шарды с разными данными. Пример:

```
example01-01-1,example01-02-1
```

## Примеры

Выборка данных с удаленного сервера:

```
SELECT * FROM remote('127.0.0.1', db.remote_engine_table) LIMIT 3;
```

Вставка данных с удаленного сервера в таблицу:

```
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;
```

## Символы подстановки в адресах

Шаблоны в фигурных скобках { } используются, чтобы сгенерировать список шардов или указать альтернативный адрес на случай отказа. В одном URL можно использовать несколько шаблонов. Поддерживаются следующие типы шаблонов.

- {*a,b*} - несколько вариантов, разделенных запятой. Весь шаблон заменяется на *a* в адресе первого шарда, заменяется на *b* в адресе второго шарда и так далее. Например, example0{1,2}-1 генерирует адреса example01-1 и example02-1.
- {*n..m*} - диапазон чисел. Этот шаблон генерирует адреса шардов с увеличивающимися индексами от *n* до *m*. example0{1..2}-1 генерирует example01-1 и example02-1.
- {*0n..0m*} - диапазон чисел с ведущими нулями. Такой вариант сохраняет ведущие нули в индексах. По шаблону example{01..03}-1 генерируются example01-1, example02-1 и example03-1.
- {*a|b*} - несколько вариантов, разделенных |. Шаблон задает адреса реплик. Например, example01-{1|2} генерирует реплики example01-1 и example01-2.

Запрос будет отправлен на первую живую реплику. При этом для remote реплики перебираются в порядке, заданном настройкой `load_balancing`.

Количество генерируемых адресов ограничено настройкой `table_function_remote_max_addresses`.

## url

Функция `url` берет данные по указанному адресу `URL` и создает из них таблицу указанной структуры со столбцами указанного формата.

Функция `url` может быть использована в запросах `SELECT` и `INSERT` с таблицами на движке `URL`.

## Синтаксис

```
url(URL, format, structure)
```

## Параметры

- URL — HTTP или HTTPS-адрес сервера, который может принимать запросы GET или POST (для запросов `SELECT` или `INSERT` соответственно). Тип: `String`.
- format — **формат** данных. Тип: `String`.
- structure — структура таблицы в формате '`UserID UInt64, Name String`'. Определяет имена и типы столбцов. Тип: `String`.

## Возвращаемое значение

Таблица с указанными форматом и структурой, а также с данными, полученными из указанного адреса `URL`.

## Примеры

Получение с HTTP-сервера первых 3 строк таблицы с данными в формате `CSV`, содержащей столбцы типа `String` и `UInt32`.

```
SELECT * FROM url('http://127.0.0.1:12345/', CSV, 'column1 String, column2 UInt32') LIMIT 3;
```

Вставка данных в таблицу:

```
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;
```

## СИМВОЛЫ ПОДСТАНОВКИ В URL

Шаблоны в фигурных скобках {} используются, чтобы сгенерировать список шардов или указать альтернативные адреса на случай отказа. Поддерживаемые типы шаблонов и примерысмотрите в описании функции `remote`.

Символ | внутри шаблонов используется, чтобы задать адреса, если предыдущие оказались недоступны. Эти адреса перебираются в том же порядке, в котором они указаны в шаблоне.

Количество адресов, которые могут быть сгенерированы, ограничено настройкой `glob_expansion_max_elements`.

## mysql

Позволяет выполнять запросы `SELECT` и `INSERT` над данными, хранящимися на удалённом MySQL сервере.

## Синтаксис

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

## Аргументы

- host:port — адрес сервера MySQL.
- database — имя базы данных на удалённом сервере.

- `table` — имя таблицы на удалённом сервере.
- `user` — пользователь MySQL.
- `password` — пароль пользователя.
- `replace_query` — флаг, отвечающий за преобразование запросов `INSERT INTO` в `REPLACE INTO`. Возможные значения:
  - `0` - выполняется запрос `INSERT INTO`.
  - `1` - выполняется запрос `REPLACE INTO`.
- `on_duplicate_clause` — выражение `ON DUPLICATE KEY on_duplicate_clause`, добавляемое в запрос `INSERT`. Может быть передано только с помощью `replace_query = 0` (если вы одновременно передадите `replace_query = 1` и `on_duplicate_clause`, будет сгенерировано исключение).

Пример: `INSERT INTO t (c1,c2) VALUES ('a', 2) ON DUPLICATE KEY UPDATE c2 = c2 + 1`, где `on\_duplicate\_clause` это `UPDATE c2 = c2 + 1`. Выражения, которые могут использоваться в качестве `on\_duplicate\_clause` в секции `ON DUPLICATE KEY`, можно посмотреть в документации по [MySQL](http://www.mysql.ru/docs/).

Простые условия `WHERE` такие как `=, !=, >, >=, <, =` выполняются на стороне сервера MySQL.

Остальные условия и ограничение выборки `LIMIT` будут выполнены в ClickHouse только после выполнения запроса к MySQL.

Поддерживает несколько реплик, которые должны быть перечислены через |. Например:

```
SELECT name FROM mysql(`mysql{1|2|3}:3306`, 'mysql_database', 'mysql_table', 'user', 'password');
```

или

```
SELECT name FROM mysql(`mysql1:3306|mysql2:3306|mysql3:3306`, 'mysql_database', 'mysql_table', 'user', 'password');
```

## Возвращаемое значение

Объект таблицы с теми же столбцами, что и в исходной таблице MySQL.

## Примечание

Чтобы отличить табличную функцию `mysql (...)` в запросе `INSERT` от имени таблицы со списком столбцов, используйте ключевые слова `FUNCTION` или `TABLE FUNCTION`. См. примеры ниже.

## Примеры

Таблица в 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  |
+-----+-----+
```

Получение данных в ClickHouse:

```
SELECT * FROM mysql('localhost:3306', 'test', 'test', 'bayonet', '123');
```

int_id	float
1	2

Замена и вставка:

```
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

## Смотрите также

- [Движок таблиц ‘MySQL’](#)
- [Использование MySQL как источника данных для внешнего словаря](#)

# postgresql

Позволяет выполнять запросы `SELECT` и `INSERT` над таблицами удаленной БД PostgreSQL.

## Синтаксис

```
postgresql('host:port', 'database', 'table', 'user', 'password'[, `schema`])
```

## Аргументы

- `host:port` — адрес сервера PostgreSQL.
- `database` — имя базы данных на удалённом сервере.
- `table` — имя таблицы на удалённом сервере.
- `user` — пользователь PostgreSQL.

- `password` — пароль пользователя.
- `schema` — имя схемы, если не используется схема по умолчанию. Необязательный аргумент.

## Возвращаемое значение

Таблица с теми же столбцами, что и в исходной таблице PostgreSQL.

## Примечание

В запросах `INSERT` для того чтобы отличить табличную функцию `postgresql(...)` от таблицы со списком имен столбцов вы должны указывать ключевые слова `FUNCTION` или `TABLE FUNCTION`. См. примеры ниже.

## Особенности реализации

Запросы `SELECT` на стороне PostgreSQL выполняются как `COPY (SELECT ...)` TO STDOUT внутри транзакции PostgreSQL только на чтение с коммитом после каждого запроса `SELECT`.

Простые условия для `WHERE` такие как `=`, `!=`, `>`, `>=`, `<`, `<=` и `IN` исполняются на стороне PostgreSQL сервера.

Все операции объединения, агрегации, сортировки, условия `IN [ array ]` и ограничения `LIMIT` выполняются на стороне ClickHouse только после того как запрос к PostgreSQL закончился.

Запросы `INSERT` на стороне PostgreSQL выполняются как `COPY "table_name" (field1, field2, ... fieldN) FROM STDIN` внутри PostgreSQL транзакции с автоматическим коммитом после каждого запроса `INSERT`.

PostgreSQL массивы конвертируются в массивы ClickHouse.

## Примечание

Будьте внимательны, в PostgreSQL массивы, созданные как `type_name[]`, являются многомерными и могут содержать в себе разное количество измерений в разных строках одной таблицы. Внутри ClickHouse допустимы только многомерные массивы с одинаковым количеством измерений во всех строках таблицы.

Поддерживает несколько реплик, которые должны быть перечислены через |. Например:

```
SELECT name FROM postgres(`postgres{1|2|3}:5432`, 'postgres_database', 'postgres_table', 'user', 'password');
```

или

```
SELECT name FROM postgres(`postgres1:5431|postgres2:5432`, 'postgres_database', 'postgres_table', 'user', 'password');
```

При использовании словаря PostgreSQL поддерживается приоритет реплик. Чем больше номер реплики, тем ниже ее приоритет. Наивысший приоритет у реплики с номером 0.

## Примеры

Таблица в 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

postgresql> SELECT * FROM test;
 int_id | int_nullable | float | str | float_nullable
-----+-----+-----+-----+
  1 |      |    2 | test |      |
(1 row)

```

Получение данных в ClickHouse:

```
SELECT * FROM postgresql('localhost:5432', 'test', 'test', 'postgresql_user', 'password') WHERE str IN ('test');
```

int_id	int_nullable	float	str	float_nullable
1	NULL	2	test	NULL

Вставка данных:

```

INSERT INTO TABLE FUNCTION postgresql('localhost:5432', 'test', 'test', 'postgresql_user', 'password') (int_id, float)
VALUES (2, 3);
SELECT * FROM postgresql('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', 'postgresql_user', 'password', 'nice.schema');

```

## См. также

- [Движок таблиц PostgreSQL](#)
- [Использование PostgreSQL как источника данных для внешнего словаря](#)

`jdbc(datasource, schema, table)` - возвращает таблицу, соединение с которой происходит через JDBC-драйвер.

Для работы этой табличной функции требуется отдельно запускать приложение [clickhouse-jdbc-bridge](#).

Данная функция поддерживает Nullable типы (на основании DDL таблицы к которой происходит запрос).

## Пример

```
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

Возвращает таблицу, подключенную через [ODBC](#).

```
odbc(connection_settings, external_database, external_table)
```

Параметры:

- `connection_settings` — название секции с настройками соединения в файле `odbc.ini`.
- `external_database` — имя базы данных во внешней СУБД.
- `external_table` — имя таблицы в `external_database`.

Чтобы использование ODBC было безопасным, ClickHouse использует отдельную программу `clickhouse-odbc-bridge`. Если драйвер ODBC подгружать непосредственно из `clickhouse-server`, то проблемы с драйвером могут привести к аварийной остановке сервера ClickHouse. ClickHouse автоматически запускает `clickhouse-odbc-bridge` по мере необходимости. Программа устанавливается из того же пакета, что и `clickhouse-server`.

Поля из внешней таблицы со значениями `NULL` получают значение по умолчанию для базового типа данных. Например, если поле в удалённой таблице MySQL имеет тип `INT NULL` оно сконвертируется в 0 (значение по умолчанию для типа данных ClickHouse `Int32`).

## Пример использования

### Получение данных из локальной установки MySQL через ODBC

Этот пример проверялся в Ubuntu Linux 18.04 для MySQL server 5.7.

Убедитесь, что unixODBC и MySQL Connector установлены.

По умолчанию (если установлен из пакетов) ClickHouse запускается от имени пользователя clickhouse. Таким образом, вам нужно создать и настроить этого пользователя на сервере MySQL.

```
$ sudo mysql
```

```
mysql> CREATE USER 'clickhouse'@'localhost' IDENTIFIED BY 'clickhouse';
mysql> GRANT ALL PRIVILEGES ON *.* TO 'clickhouse'@'clickhouse' WITH GRANT OPTION;
```

Теперь настроим соединение в /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
```

Вы можете проверить соединение с помощью утилиты isql из установки unixODBC.

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

Таблица в 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)
```

Получение данных из таблицы MySQL в ClickHouse:

```
SELECT * FROM odbc('DSN=mysqlconn', 'test', 'test')
```

int_id	int_nullable	float	float_nullable
1	0	2	0

Смотрите также

- Внешние словари ODBC
- Движок таблиц ODBC.

## hdfs

Создаёт таблицу из файла в HDFS. Данная табличная функция похожа на табличные функции [url](#) и [file](#).

```
hdfs(URI, format, structure)
```

### Входные параметры

- `URI` — URI файла в HDFS. Путь к файлу поддерживает следующие шаблоны в режиме доступа только для чтения `*`, `?`, `{abc,def}` и `{N..M}`, где `N, M` — числа, `'abc', 'def'` — строки.
- `format` — [формат](#) файла.
- `structure` — структура таблицы. Формат `'column1_name column1_type, column2_name column2_type, ...'`.

### Возвращаемое значение

Таблица с указанной структурой, предназначенная для чтения или записи данных в указанном файле.

### Пример

Таблица из `hdfs://hdfs1:9000/test` и выборка первых двух строк из неё:

```
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

### Шаблоны в пути

- `*` — Заменяет любое количество любых символов кроме `/`, включая отсутствие символов.
- `?` — Заменяет ровно один любой символ.
- `{some_string,another_string,yet_another_one}` — Заменяет любую из строк `'some_string', 'another_string', 'yet_another_one'`.
- `{N..M}` — Заменяет любое число в интервале от `N` до `M` включительно (может содержать ведущие нули).

Конструкция с `{}` аналогична табличной функции [remote](#).

### Warning

Если ваш список файлов содержит интервал с ведущими нулями, используйте конструкцию с фигурными скобками для каждой цифры по отдельности или используйте `?`.

Шаблоны могут содержаться в разных частях пути. Обрабатываться будут ровно те файлы, которые и удовлетворяют всему шаблону пути, и существуют в файловой системе.

## Виртуальные столбцы

- `_path` — Путь к файлу.
- `_file` — Имя файла.

### Смотрите также

- [Виртуальные столбцы](#)

## Табличная Функция S3

Предоставляет табличный интерфейс для выбора/вставки файлов в [Amazon S3](#). Эта табличная функция похожа на [hdfs](#), но обеспечивает специфические для S3 возможности.

### Синтаксис

```
s3(path, [aws_access_key_id, aws_secret_access_key,] format, structure, [compression])
```

### Аргументы

- `path` — URL-адрес бакета с указанием пути к файлу. Поддерживает следующие подстановочные знаки в режиме "только чтение": `*`, `?`, `{abc,def}` и `{N..M}` где `N, M` — числа, `'abc', 'def'` — строки.  
Подробнее смотри [здесь](#).
- `format` — [формат](#) файла.
- `structure` — структура таблицы. Формат `'column1_name column1_type, column2_name column2_type, ...'`.
- `compression` — автоматически обнаруживает сжатие по расширению файла. Возможные значения: `none, gzip/gz, brotli/br, xz/LZMA, zstd/zst`. Необязательный параметр.

### Возвращаемые значения

Таблица с указанной структурой для чтения или записи данных в указанный файл.

### Примеры

Создание таблицы из файла S3 `https://storage.yandexcloud.net/my-test-bucket-768/data.csv` и выбор первых трех столбцов из нее:

Запрос:

```
SELECT *
FROM s3('https://storage.yandexcloud.net/my-test-bucket-768/data.csv', 'CSV', 'column1 UInt32, column2 UInt32,
column3 UInt32')
LIMIT 2;
```

Результат:

column1	column2	column3
1	2	3
3	2	1

То же самое, но файл со сжатием gzip:

Запрос:

```
SELECT *
FROM s3('https://storage.yandexcloud.net/my-test-bucket-768/data.csv.gz', 'CSV', 'column1 UInt32, column2 UInt32,
column3 UInt32', 'gzip')
LIMIT 2;
```

Результат:

column1	column2	column3
1	2	3
3	2	1

## Примеры использования

Предположим, у нас есть несколько файлов со следующими URI на 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)'

Подсчитаем количество строк в файлах, заканчивающихся цифрами от 1 до 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

Подсчитаем общее количество строк во всех файлах этих двух каталогов:

```
SELECT count(*)
FROM s3('https://storage.yandexcloud.net/my-test-bucket-768/{some,another}_prefix/*', 'CSV', 'name String, value
UInt32');
```

count()
24

Warning

Если список файлов содержит диапазоны чисел с ведущими нулями, используйте конструкцию с фигурными скобками для каждой цифры отдельно или используйте ?.

Подсчитаем общее количество строк в файлах с именами 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 |
```

Запишем данные в файл 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);
```

Запишем данные из существующей таблицы в файл 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')  
SELECT name, value FROM existing_table;
```

## Партиционирование при записи данных

Если при добавлении данных в таблицу S3 указать выражение PARTITION BY, то для каждого значения ключа партиционирования создается отдельный файл. Это повышает эффективность операций чтения.

### Примеры

1. При использовании ID партиции в имени ключа создаются отдельные файлы:

```
INSERT INTO TABLE FUNCTION  
s3('http://bucket.amazonaws.com/my_bucket/file_{_partition_id}.csv', 'CSV', 'a UInt32, 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);
```

В результате данные будут записаны в три файла: file\_x.csv, file\_y.csv и file\_z.csv.

2. При использовании ID партиции в названии бакета создаются файлы в разных бакетах:

```
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);
```

В результате будут созданы три файла в разных бакетах: my\_bucket\_1/file.csv, my\_bucket\_10/file.csv и my\_bucket\_20/file.csv.

### Смотрите также

- [Движок таблиц S3](#)

# input

`input(structure)` - табличная функция, позволяющая эффективно преобразовывать и вставлять отправленные на сервер данные, имеющие структуру `structure`, в таблицу другой структуры.

`structure` - структура отправляемых на сервер данных в формате '`column1_name column1_type, column2_name column2_type, ...`'.

Например: '`id UInt32, name String`'.

Данная функция может быть использована только в запросе `INSERT SELECT` и только один раз, но в остальном ведет себя как обычная табличная функция (можно указать в подзапросе и т.д.).

Данные можно отправлять любым стандартным способом как для обычного `INSERT` запроса и в любом доступном **формате**, который указывается в конце запроса (в отличие от обычного `INSERT SELECT`).

Главная особенность данной функции в том, что сервер при получении данных от клиента одновременно преобразует их в соответствии со списком выражений в `SELECT` части и вставляет в целевую таблицу. Временная таблица со всеми переданными данными не создается.

## Примеры

- Пусть у таблицы `test` следующая структура (`a String, b String`), а в файле `data.csv` данные имеют другую структуру (`col1 String, col2 Date, col3 Int32`). Запрос для вставки данных из файла `data.csv` в таблицу `test` с одновременным преобразованием и использованием функций выглядит так:

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

- Если в `data.csv` лежат данные той же структуры `test_structure`, что и у таблицы `test`, то следующие два запроса эквивалентны:

```
$ 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

Генерирует случайные данные с заданной схемой.

Позволяет заполнять тестовые таблицы данными.

Поддерживает все типы данных, которые могут храниться в таблице, за исключением `LowCardinality` и `AggregateFunction`.

```
generateRandom('name TypeName[, name TypeName]...', [, 'random_seed'][, 'max_string_length'][, 'max_array_length']])
```

## Аргументы

- `name` — название соответствующего столбца.

- `TypeName` — тип соответствующего столбца.
- `max_array_length` — максимальная длина массива для всех сгенерированных массивов. По умолчанию `10`.
- `max_string_length` — максимальная длина строки для всех генерируемых строк. По умолчанию `10`.
- `random_seed` — укажите состояние генератора случайных чисел вручную, чтобы получить стабильные результаты. Если значение равно `NULL` - генератор инициализируется случайным состоянием.

## Возвращаемое значение

Объект таблицы с запрошенной схемой.

## Пример

```
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

Позволяет обратиться ко всем шардам существующего кластера, который сконфигурирован в секции `remote_servers` без создания таблицы типа **Distributed**. В запросе используется одна реплика каждого шарда.

Функция `clusterAllReplicas` работает также как `cluster`, но каждая реплика в кластере используется как отдельный шард/отдельное соединение.

### Примечание

Все доступные кластеры перечислены в таблице **system.clusters**.

### Синтаксис

```
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])
```

### Аргументы

- `cluster_name` – имя кластера, который обозначает подмножество адресов и параметров подключения к удаленным и локальным серверам, входящим в кластер.
- `db.table` или `db, table` - имя базы данных и таблицы.
- `sharding_key` - ключ шардирования. Необязательный аргумент. Указывается, если данные добавляются более чем в один шард кластера.

## Возвращаемое значение

Набор данных из кластеров.

## Использование макросов

cluster\_name может содержать макрос — подстановку в фигурных скобках. Эта подстановка заменяется на соответствующее значение из секции **macros** конфигурационного файла .

Пример:

```
SELECT * FROM cluster('{cluster}', default.example_table);
```

## Использование и рекомендации

Использование табличных функций `cluster` и `clusterAllReplicas` менее оптимально, чем создание таблицы типа `Distributed`, поскольку в этом случае при каждом новом запросе устанавливается новое соединение с сервером. При обработке большого количества запросов всегда создавайте `Distributed` таблицу заранее и не используйте табличные функции `cluster` и `clusterAllReplicas`.

Табличные функции `cluster` and `clusterAllReplicas` могут быть полезны в следующих случаях:

- Чтение данных из конкретного кластера для сравнения данных, отладки и тестирования.
- Запросы к разным ClickHouse кластерам и репликам в целях исследования.
- Нечастых распределенных запросов которые делаются вручную.

Настройки соединения `user`, `password`, `host`, `port`, `compression`, `secure` берутся из секции `<remote_servers>` файлов конфигурации. См. подробности в разделе **Distributed**

## См. также

- [skip\\_unavailable\\_shards](#)
- [load\\_balancing](#)

# null

Создает временную таблицу указанной структуры с движком `Null`. В соответствии со свойствами движка, данные в таблице игнорируются, а сама таблица удаляется сразу после выполнения запроса. Функция используется для удобства написания тестов и демонстрационных примеров.

## Синтаксис

```
null('structure')
```

## Параметр

- `structure` — список колонок и их типов. [String](#).

## Возвращаемое значение

Временная таблица указанной структуры с движком `Null`.

## Пример

Один запрос с функцией `null`:

```
INSERT INTO function null('x UInt64') SELECT * FROM numbers_mt(1000000000);
```

заменяет три запроса:

```
CREATE TABLE t (x UInt64) ENGINE = Null;
INSERT INTO t SELECT * FROM numbers_mt(1000000000);
DROP TABLE IF EXISTS t;
```

См. также:

- [Движок таблиц Null](#)

## dictionary

Отображает данные [словаря](#) как таблицу ClickHouse. Работает аналогично движку [Dictionary](#).

### Синтаксис

```
dictionary('dict')
```

### Аргументы

- `dict` — имя словаря. [String](#).

### Возвращаемое значение

Таблица ClickHouse.

### Пример

Входная таблица `dictionary_source_table`:

id	value
0	0
1	1

Создаем словарь:

```
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());
```

Запрос:

```
SELECT * FROM dictionary('new_dictionary');
```

Результат:

id	value
0	0
1	1

### Смотрите также

- [Движок Dictionary](#)

# Табличная функция s3Cluster

Позволяет обрабатывать файлы из [Amazon S3](#) параллельно из многих узлов в указанном кластере. На узле-инициаторе функция создает соединение со всеми узлами в кластере, заменяет символы '\*' в пути к файлу S3 и динамически отправляет каждый файл. На рабочем узле функция запрашивает у инициатора следующую задачу и обрабатывает ее. Это повторяется до тех пор, пока все задачи не будут завершены.

## Синтаксис

```
s3Cluster(cluster_name, source, [access_key_id, secret_access_key,] format, structure)
```

## Аргументы

- `cluster_name` — имя кластера, используемое для создания набора адресов и параметров подключения к удаленным и локальным серверам.
- `source` — URL файла или нескольких файлов. Поддерживает следующие символы подстановки: `*`, `?`, `{'abc','def'}` и `{N..M}`, где N, M — числа, abc, def — строки. Подробнее смотрите в разделе [Символы подстановки](#).
- `access_key_id` и `secret_access_key` — ключи, указывающие на учетные данные для использования с точкой приема запроса. Необязательные параметры.
- `format` — [формат](#) файла.
- `structure` — структура таблицы. Формат '`column1_name column1_type, column2_name column2_type, ...!`'.

## Возвращаемое значение

Таблица с указанной структурой для чтения или записи данных в указанный файл.

## Примеры

Вывод данных из всех файлов кластера `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);
```

Подсчет общего количества строк во всех файлах кластера `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)))');
```

## Внимание

Если список файлов содержит диапазоны чисел с ведущими нулями, используйте конструкцию с фигурными скобками для каждой цифры отдельно или используйте `?`.

## Смотрите также

- [Движок таблиц S3](#)
- [Табличная функция s3](#)

# sqlite

Позволяет выполнять запросы к данным, хранящимся в базе данных [SQLite](#).

## Синтаксис

```
sqlite('db_path', 'table_name')
```

## Аргументы

- `db_path` — путь к файлу с базой данных SQLite. [String](#).
- `table_name` — имя таблицы в базе данных SQLite. [String](#).

## Возвращаемое значение

- Объект таблицы с теми же столбцами, что и в исходной таблице SQLite.

## Пример

Запрос:

```
SELECT * FROM sqlite('sqlite.db', 'table1') ORDER BY col2;
```

Результат:

col1	col2
line1	1
line2	2
line3	3

## См. также

- [SQLite](#) движок таблиц

# view

Преобразовывает подзапрос в таблицу. Функция реализовывает представления (смотрите [CREATE VIEW](#)). Результирующая таблица не хранит данные, а только сохраняет указанный запрос SELECT. При чтении из таблицы, ClickHouse выполняет запрос и удаляет все ненужные столбцы из результата.

## Синтаксис

```
view(subquery)
```

## Аргументы

- `subquery` — запрос `SELECT`.

## Возвращаемое значение

- Таблица.

## Пример

Входная таблица:

	id	name	days
1	January	31	
2	February	29	
3	March	31	
4	April	30	

Запрос:

```
SELECT * FROM view(SELECT name FROM months);
```

Результат:

name
January
February
March
April

Вы можете использовать функцию `view` как параметр табличных функций `remote` и `cluster`:

```
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));
```

## Смотрите также

- [view](#)

## Словари

Словарь — это отображение (ключ -> атрибуты), которое удобно использовать для различного вида справочников.

ClickHouse поддерживает специальные функции для работы со словарями, которые можно использовать в запросах. Проще и эффективнее использовать словари с помощью функций, чем `JOIN` с таблицами-справочниками.

ClickHouse поддерживает:

- [Встроенные словари](#) со специфическим [набором функций](#).
- [Подключаемые \(внешние\) словари](#) с [набором функций](#).

## Внешние словари

Существует возможность подключать собственные словари из различных источников данных. Источником данных для словаря может быть локальный текстовый/исполнимый файл, HTTP(s) ресурс или другая СУБД. Подробнее смотрите в разделе [«Источники внешних словарей»](#).

## ClickHouse:

- Полностью или частично хранит словари в оперативной памяти.
- Периодически обновляет их и динамически подгружает отсутствующие значения.
- Позволяет создавать внешние словари с помощью xml-файлов или [DDL-запросов](#).

Конфигурация внешних словарей может находиться в одном или нескольких xml-файлах. Путь к конфигурации указывается в параметре [dictionaries\\_config](#).

Словари могут загружаться при старте сервера или при первом использовании, в зависимости от настройки [dictionaries\\_lazy\\_load](#).

Системная таблица [system.dictionaries](#) содержит информацию о словарях, сконфигурированных на сервере. Для каждого словаря там можно найти:

- Статус словаря.
- Конфигурационные параметры.
- Метрики, наподобие количества занятой словарём RAM или количества запросов к словарю с момента его успешной загрузки.

Конфигурационный файл словарей имеет вид:

```
<clickhouse>
  <comment>Необязательный элемент с любым содержимым. Игнорируется сервером ClickHouse.</comment>

  <!--Необязательный элемент, имя файла с подстановками-->
  <include_from>/etc/metrika.xml</include_from>

  <dictionary>
    <!-- Конфигурация словаря -->
  </dictionary>

  ...
  <dictionary>
    <!-- Конфигурация словаря -->
  </dictionary>
</clickhouse>
```

В одном файле можно [сконфигурировать](#) произвольное количество словарей.

Если вы создаёте внешние словари [DDL-запросами](#), то не задавайте конфигурацию словаря в конфигурации сервера.

## Внимание

Можно преобразовывать значения по небольшому словарю, описав его в запросе [SELECT](#) (см. функцию [transform](#)). Эта функциональность не связана с внешними словарями.

## Смотрите также

- [Настройка внешнего словаря](#)
- [Хранение словарей в памяти](#)
- [Обновление словарей](#)
- [Источники внешних словарей](#)

- Ключ и поля словаря
- Функции для работы с внешними словарями

## Настройка внешнего словаря

XML-конфигурация словаря имеет следующую структуру:

```
<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>
```

Соответствующий **DDL-запрос** имеет следующий вид:

```
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** — Идентификатор, под которым словарь будет доступен для использования. Используйте символы [a-zA-Z0-9\\_].
- **source** — Источник словаря.
- **layout** — Размещение словаря в памяти.
- **structure** — Структура словаря. Ключ и атрибуты, которые можно получить по ключу.
- **lifetime** — Периодичность обновления словарей.

## Хранение словарей в памяти

Словари можно размещать в памяти множеством способов.

Рекомендуем **flat**, **hashed** и **complex\_key\_hashed**. Скорость обработки словарей при этом максимальна.

Размещение с кэшированием не рекомендуется использовать из-за потенциально низкой производительности и сложностей в подборе оптимальных параметров. Читайте об этом подробнее в разделе **cache**.

Повысить производительность словарей можно следующими способами:

- Вызывать функцию для работы со словарём после `GROUP BY`.
- Помечать извлекаемые атрибуты как инъективные. Атрибут называется инъективным, если разным ключам соответствуют разные значения атрибута. Тогда при использовании в `GROUP BY` функции, достающей значение атрибута по ключу, эта функция автоматически выносится из `GROUP BY`.

При ошибках работы со словарями ClickHouse генерирует исключения. Например, в следующих ситуациях:

- При обращении к словарю, который не удалось загрузить.
- При ошибке запроса к `cached`-словарю.

Список внешних словарей и их статус можно посмотреть в таблице `system.dictionaries`.

Общий вид конфигурации:

```
<clickhouse>
  <dictionary>
    ...
    <layout>
      <layout_type>
        <!-- layout settings -->
      </layout_type>
    </layout>
  ...
</dictionary>
</clickhouse>
```

Соответствующий **DDL-запрос**:

```
CREATE DICTIONARY (...)

...
LAYOUT(LAYOUT_TYPE(param value)) -- layout settings
...
```

## Способы размещения словарей в памяти

- `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\\_ssdcache](#)
- [direct](#)
- [complex\\_key\\_direct](#)
- [ip\\_trie](#)

## flat

Словарь полностью хранится в оперативной памяти в виде плоских массивов. Объём памяти, занимаемой словарём, пропорционален размеру самого большого ключа (по объему).

Ключ словаря имеет тип [UInt64](#) и его величина ограничена параметром `max_array_size` (значение по умолчанию — 500 000). Если при создании словаря обнаружен ключ больше, то ClickHouse бросает исключение и не создает словарь. Начальный размер плоских массивов словарей контролируется параметром `initial_array_size` (по умолчанию - 1024).

Поддерживаются все виды источников. При обновлении данные (из файла или из таблицы)читываются целиком.

Это метод обеспечивает максимальную производительность среди всех доступных способов размещения словаря.

Пример конфигурации:

```
<layout>
<flat>
  <initial_array_size>50000</initial_array_size>
  <max_array_size>5000000</max_array_size>
</flat>
</layout>
```

или

```
LAYOUT(FLAT(INITIAL_ARRAY_SIZE 50000 MAX_ARRAY_SIZE 5000000))
```

## hashed

Словарь полностью хранится в оперативной памяти в виде хэш-таблиц. Словарь может содержать произвольное количество элементов с произвольными идентификаторами. На практике количество ключей может достигать десятков миллионов элементов.

Если `preallocate` имеет значение `true` (по умолчанию `false`), хэш-таблица будет предварительно определена (это ускорит загрузку словаря). Используйте этот метод только в случае, если:

- Источник поддерживает произвольное количество элементов (пока поддерживается только источником [ClickHouse](#)).
- В данных нет дубликатов (иначе это может увеличить объем используемой памяти хэш-таблицы).

Поддерживаются все виды источников. При обновлении данные (из файла, из таблицы) читаются целиком.

Пример конфигурации:

```
<layout>
  <hashed>
    <preallocate>0</preallocate>
  </hashed>
</layout>
```

или

```
LAYOUT(HASHED(PREALLOCATE 0))
```

## sparse\_hashed

Аналогичен `hashed`, но при этом занимает меньше места в памяти и генерирует более высокую загрузку CPU.

Для этого типа размещения также можно задать `preallocate` в значении `true`. В данном случае это более важно, чем для типа `hashed`.

Пример конфигурации:

```
<layout>
  <sparse_hashed />
</layout>
```

или

```
LAYOUT(SPARSE_HASHED([PREALLOCATE 0]))
```

## complex\_key\_hashed

Тип размещения предназначен для использования с составными [ключами](#). Аналогичен `hashed`.

Пример конфигурации:

```
<layout>
  <complex_key_hashed />
</layout>
```

или

```
LAYOUT(COMPLEX_KEY_HASHED())
```

## complex\_key\_sparse\_hashed

Тип размещения предназначен для использования с составными [ключами](#). Аналогичен `sparse_hashed`.

Пример конфигурации:

```
<layout>
  <complex_key_sparse_hashed />
</layout>
```

или

```
LAYOUT(COMPLEX_KEY_SPARSE_HASHED())
```

## hashed\_array

Словарь полностью хранится в оперативной памяти. Каждый атрибут хранится в массиве. Ключевой атрибут хранится в виде хеш-таблицы, где его значение является индексом в массиве атрибутов. Словарь может содержать произвольное количество элементов с произвольными идентификаторами. На практике количество ключей может достигать десятков миллионов элементов.

Поддерживаются все виды источников. При обновлении данные (из файла, из таблицы) считаются целиком.

Пример конфигурации:

```
<layout>
  <hashed_array>
  </hashed_array>
</layout>
```

или

```
LAYOUT(HASHED_ARRAY())
```

## complex\_key\_hashed\_array

Тип размещения предназначен для использования с составными **ключами**. Аналогичен [hashed\\_array](#).

Пример конфигурации:

```
<layout>
  <complex_key_hashed_array />
</layout>
```

или

```
LAYOUT(COMPLEX_KEY_HASHED_ARRAY())
```

## range\_hashed

Словарь хранится в оперативной памяти в виде хеш-таблицы с упорядоченным массивом диапазонов и соответствующих им значений.

Этот способ размещения работает также как и `hashed` и позволяет дополнительно к ключу использовать диапазоны по дате/времени (произвольному числовому типу).

Пример: таблица содержит скидки для каждого рекламодателя в виде:

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

Чтобы использовать выборку по диапазонам дат, необходимо в **structure** определить элементы `range_min`, `range_max`. В этих элементах должны присутствовать элементы `name` и `type` (если `type` не указан, будет использован тип по умолчанию – Date). `type` может быть любым численным типом (Date/DateTime/UInt64/Int32/др.).

Пример:

```
<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>
  ...
  ...
```

ИЛИ

```
CREATE DICTIONARY somedict (
  id UInt64,
  first Date,
  last Date
)
PRIMARY KEY id
LAYOUT(RANGE_HASHED())
RANGE(MIN first MAX last)
```

Для работы с такими словарями в функцию `dictGetT` необходимо передавать дополнительный аргумент, для которого подбирается диапазон:

```
dictGetT('dict_name', 'attr_name', id, date)
```

Функция возвращает значение для заданных `id` и диапазона дат, в который входит переданная дата.

Особенности алгоритма:

- Если не найден `id` или для найденного `id` не найден диапазон, то возвращается значение по умолчанию для словаря.
- Если есть перекрывающиеся диапазоны, то возвращается значение из любого (случайного) подходящего диапазона.
- Если граница диапазона `NULL` или некорректная дата (1900-01-01), то диапазон считается открытым. Диапазон может быть открытым с обеих сторон.

Пример конфигурации:

```

<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>

```

ИЛИ

```

CREATE DICTIONARY somedict(
  Abcdef UInt64,
  StartTimeStamp UInt64,
  EndTimeStamp UInt64,
  XXXType String DEFAULT ''
)
PRIMARY KEY Abcdef
RANGE(MIN StartTimeStamp MAX EndTimeStamp)

```

## complex\_key\_range\_hashed

Словарь хранится в оперативной памяти в виде хэш-таблицы с упорядоченным массивом диапазонов и соответствующих им значений (см. [range\\_hashed](#)). Данный тип размещения предназначен для использования с составными [ключами](#).

Пример конфигурации:

```

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

Словарь хранится в кэше, состоящем из фиксированного количества ячеек. Ячейки содержат часто используемые элементы.

При поиске в словаре сначала просматривается кэш. На каждый блок данных, все не найденные в кэше или устаревшие ключи запрашиваются у источника с помощью `SELECT attrs... FROM db.table WHERE id IN (k1, k2, ...)`. Затем, полученные данные записываются в кэш.

Если ключи не были найдены в словаре, то для обновления кэша создается задание и добавляется в очередь обновлений. Параметры очереди обновлений можно устанавливать настройками `max_update_queue_size`, `update_queue_push_timeout_milliseconds`, `query_wait_timeout_milliseconds`, `max_threads_for_updates`

Для cache-словарей при помощи настройки `allow_read_expired_keys` может быть задано время устаревания `lifetime` данных в кэше. Если с момента загрузки данных в ячейку прошло больше времени, чем `lifetime`, то значение не используется, а ключ устаревает. Ключ будет запрошен заново при следующей необходимости его использовать.

Это наименее эффективный из всех способов размещения словарей. Скорость работы кэша очень сильно зависит от правильности настройки и сценария использования. Словарь типа `cache` показывает высокую производительность лишь при достаточно большой частоте успешных обращений (рекомендуется 99% и выше). Посмотреть среднюю частоту успешных обращений (`hit rate`) можно в таблице [system.dictionaries](#).

Если параметр `allow_read_expired_keys` выставлен в 1 (0 по умолчанию), то словарь поддерживает асинхронные обновления. Если клиент запрашивает ключи, которые находятся в кэше, но при этом некоторые из них устарели, то словарь вернет устаревшие ключи клиенту и запросит их асинхронно у источника.

Чтобы увеличить производительность кэша, используйте подзапрос с `LIMIT`, а снаружи вызывайте функцию со словарём.

Поддерживаются [источники](#): MySQL, ClickHouse, executable, HTTP.

Пример настройки:

```
<layout>
  <cache>
    <!-- Размер кэша в количестве ячеек. Округляется вверх до степени двух. -->
    <size_in_cells>1000000000</size_in_cells>
    <!-- Позволить читать устаревшие ключи. -->
    <allow_read_expired_keys>0</allow_read_expired_keys>
    <!-- Максимальный размер очереди обновлений. -->
    <max_update_queue_size>100000</max_update_queue_size>
    <!-- Максимальное время (в миллисекундах) для отправки в очередь. -->
    <update_queue_push_timeout_milliseconds>10</update_queue_push_timeout_milliseconds>
    <!-- Максимальное время ожидания (в миллисекундах) для выполнения обновлений. -->
    <query_wait_timeout_milliseconds>60000</query_wait_timeout_milliseconds>
    <!-- Максимальное число потоков для обновления кэша словаря. -->
    <max_threads_for_updates>4</max_threads_for_updates>
  </cache>
</layout>
```

или

```
LAYOUT(CACHE(SIZE_IN_CELLS 1000000000))
```

Укажите достаточно большой размер кэша. Количество ячеек следует подобрать экспериментальным путём:

1. Выставить некоторое значение.

2. Запросами добиться полной заполненности кэша.
3. Оценить потребление оперативной памяти с помощью таблицы `system.dictionaries`.
4. Увеличивать/уменьшать количество ячеек до получения требуемого расхода оперативной памяти.

## Warning

Не используйте в качестве источника ClickHouse, поскольку он медленно обрабатывает запросы со случайным чтением.

## complex\_key\_cache

Тип размещения предназначен для использования с составными [ключами](#). Аналогичен `cache`.

### ssd\_cache

Похож на `cache`, но хранит данные на SSD, а индекс в оперативной памяти. Все параметры, относящиеся к очереди обновлений, могут также быть применены к SSD-кэш словарям.

```
<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>
```

или

```
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

Тип размещения предназначен для использования с составными [ключами](#). Похож на `ssd_cache`.

### direct

Словарь не хранит данные локально и взаимодействует с источником непосредственно в момент запроса.

Ключ словаря имеет тип `UInt64`.

Поддерживаются все виды [источников](#), кроме локальных файлов.

Пример конфигурации:

```
<layout>
  <direct />
</layout>
```

или

```
LAYOUT(DIRECT())
```

## complex\_key\_direct

Тип размещения предназначен для использования с составными **ключами**. Аналогичен `direct`.

### ip\_trie

Тип размещения предназначен для сопоставления префиксов сети (IP адресов) с метаданными, такими как ASN.

Пример: таблица содержит префиксы сети и соответствующие им номера AS и коды стран:

prefix	asn	ccs2
202.79.32.0/20	17501	NP
2620:0:870::/48	3856	US
2a02:6b8:1::/48	13238	RU
2001:db8::/32	65536	ZZ

При использовании такого макета структура должна иметь составной ключ.

Пример:

```
<structure>
  <key>
    <attribute>
      <name>prefix</name>
      <type>String</type>
    </attribute>
  </key>
  <attribute>
    <name>asn</name>
    <type>UInt32</type>
    <null_value />
  </attribute>
  <attribute>
    <name>ccs2</name>
    <type>String</type>
    <null_value>??</null_value>
  </attribute>
  ...
</structure>
<layout>
  <ip_trie>
    <!-- Ключевой атрибут `prefix` будет доступен через dictGetString -->
    <!-- Эта опция увеличивает потребляемую память -->
    <access_to_key_from_attributes>true</access_to_key_from_attributes>
  </ip_trie>
</layout>
```

ИЛИ

```
CREATE DICTIONARY somedict (
  prefix String,
  asn UInt32,
  ccs2 String DEFAULT '??'
)
PRIMARY KEY prefix
```

Этот ключ должен иметь только один атрибут типа `String`, содержащий допустимый префикс IP. Другие типы еще не поддерживаются.

Для запросов необходимо использовать те же функции (`dictGetT` с кортежем), что и для словарей с составными ключами:

```
dictGetT('dict_name', 'attr_name', tuple(ip))
```

Функция принимает либо `UInt32` для IPv4, либо `FixedString(16)` для IPv6:

```
dictGetString('prefix', 'asn', tuple(IPv6StringToNum('2001:db8::1')))
```

Некакие другие типы не поддерживаются. Функция возвращает атрибут для префикса, соответствующего данному IP-адресу. Если есть перекрывающиеся префиксы, возвращается наиболее специфический.

Данные должны полностью помещаться в оперативной памяти.

## Обновление словарей

ClickHouse периодически обновляет словари. Интервал обновления для полностью загружаемых словарей и интервал инвалидации для кэшируемых словарей определяется в теге `<lifetime>` в секундах.

Обновление словарей (кроме загрузки при первом использовании) не блокирует запросы - во время обновления используется старая версия словаря. Если при обновлении возникнет ошибка, то ошибка пишется в лог сервера, а запросы продолжат использовать старую версию словарей.

Пример настройки:

```
<dictionary>
...
<lifetime>300</lifetime>
...
</dictionary>
```

ИЛИ

```
CREATE DICTIONARY (...)

...
LIFETIME(300)
...
```

Настройка `<lifetime>0</lifetime>` (`LIFETIME(0)`) запрещает обновление словарей.

Можно задать интервал, внутри которого ClickHouse равномерно-случайно выберет время для обновления. Это необходимо для распределения нагрузки на источник словаря при обновлении на большом количестве серверов.

Пример настройки:

```
<dictionary>
...
<lifetime>
  <min>300</min>
  <max>360</max>
</lifetime>
...
</dictionary>
```

или

```
LIFETIME(MIN 300 MAX 360)
```

Если `<min>0</min>` и `<max>0</max>`, ClickHouse не перезагружает словарь по истечении времени. В этом случае ClickHouse может перезагрузить данные словаря, если изменился XML файл с конфигурацией словаря или если была выполнена команда `SYSTEM RELOAD DICTIONARY`.

При обновлении словарей сервер ClickHouse применяет различную логику в зависимости от типа **источника**:

- У текстового файла проверяется время модификации. Если время изменилось по отношению к запомненному ранее, то словарь обновляется.
- Для MySQL источника время модификации проверяется запросом `SHOW TABLE STATUS` (для MySQL 8 необходимо отключить кеширование мета-информации в MySQL `set global information_schema_stats_expiry=0`).
- Словари из других источников по умолчанию обновляются каждый раз.

Для других источников (ODBC, PostgreSQL, ClickHouse и т.д.) можно настроить запрос, который позволит обновлять словари только в случае их фактического изменения, а не каждый раз. Чтобы это сделать необходимо выполнить следующие условия/действия:

- В таблице словаря должно быть поле, которое гарантированно изменяется при обновлении данных в источнике.
- В настройках источника указывается запрос, который получает изменяющееся поле. Результат запроса сервер ClickHouse интерпретирует как строку и если эта строка изменилась по отношению к предыдущему состоянию, то словарь обновляется. Запрос следует указывать в поле `<invalidate_query>` настроек **источника**.

Пример настройки:

```
<dictionary>
...
<odbc>
  ...
  <invalidate_query>SELECT update_time FROM dictionary_source where id = 1</invalidate_query>
</odbc>
...
</dictionary>
```

или

```
...
SOURCE(ODBC(... invalidate_query 'SELECT update_time FROM dictionary_source where id = 1'))
...
```

Для словарей `Cache`, `ComplexKeyCache`, `SSDCache` и `SSDComplexKeyCache` поддерживается как синхронное, так и асинхронное обновление.

Словари `Flat`, `Hashed` и `ComplexKeyHashed` могут запрашивать только те данные, которые были изменены после предыдущего обновления. Если `update_field` указано как часть конфигурации источника словаря, к запросу данных будет добавлено время предыдущего обновления в секундах. В зависимости от типа источника (`Executable`, `HTTP`, `MySQL`, `PostgreSQL`, `ClickHouse`, `ODBC`) к `update_field` будет применена соответствующая логика перед запросом данных из внешнего источника.

- Если источник `HTTP`, то `update_field` будет добавлено в качестве параметра запроса, а время последнего обновления — в качестве значения параметра.
- Если источник `Executable`, то `update_field` будет добавлено в качестве аргумента исполняемого скрипта, время последнего обновления — в качестве значения аргумента.
- Если источник `ClickHouse`, `MySQL`, `PostgreSQL` или `ODBC`, то будет дополнительная часть запроса `WHERE`, где `update_field` будет больше или равно времени последнего обновления.

Если установлена опция `update_field`, то может быть установлена дополнительная опция `update_lag`. Значение `update_lag` вычитается из времени предыдущего обновления перед запросом обновленных данных.

Пример настройки:

```
<dictionary>
...
  <clickhouse>
    ...
      <update_field>added_time</update_field>
      <update_lag>15</update_lag>
    </clickhouse>
  ...
</dictionary>
```

или

```
...
  SOURCE(CLICKHOUSE(... update_field 'added_time' update_lag 15))
...
```

## Источники внешних словарей

Внешний словарь можно подключить из множества источников.

Общий вид XML-конфигурации:

```
<clickhouse>
  <dictionary>
    ...
      <source>
        <source_type>
          <!-- Source configuration -->
        </source_type>
      </source>
    ...
  </dictionary>
  ...
</clickhouse>
```

Аналогичный DDL-запрос:

```
CREATE DICTIONARY dict_name (...)  
...  
SOURCE(SOURCE_TYPE(param1 val1 ... paramN valN)) -- Source configuration  
...
```

Источник настраивается в разделе source.

Для типов источников [Локальный файл](#), [Исполняемый файл](#), [HTTP\(s\)](#), [ClickHouse](#) доступны дополнительные настройки:

```
<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>
```

или

```
SOURCE(FILE(path './user_files/os.tsv' format 'TabSeparated'))  
SETTINGS(format_csv_allow_single_quotes = 0)
```

Типы источников (source\_type):

- [Локальный файл](#)
- [Исполняемый файл](#)
- [Исполняемый пул](#)
- [HTTP\(s\)](#)
- СУБД:
  - [ODBC](#)
  - [MySQL](#)
  - [PostgreSQL](#)
  - [ClickHouse](#)
  - [MongoDB](#)
  - [Redis](#)
  - [Cassandra](#)
  - [PostgreSQL](#)

## Локальный файл

Пример настройки:

```
<source>
<file>
  <path>/opt/dictionaries/os.tsv</path>
  <format>TabSeparated</format>
</file>
</source>
```

ИЛИ

```
SOURCE(FILE(path './user_files/os.tsv' format 'TabSeparated'))
```

Поля настройки:

- `path` — абсолютный путь к файлу.
- `format` — формат файла. Поддерживаются все форматы, описанные в разделе [Форматы](#).

Если словарь с источником `FILE` создается с помощью DDL-команды (`CREATE DICTIONARY ...`), источник словаря должен быть расположен в каталоге `user_files`. Иначе пользователи базы данных будут иметь доступ к произвольному файлу на узле ClickHouse.

#### Смотрите также

- [Функция dictionary](#)

## Исполняемый файл

Работа с исполняемым файлом зависит от [размещения словаря в памяти](#). Если тип размещения словаря `cache` и `complex_key_cache`, то ClickHouse запрашивает необходимые ключи, отправляя запрос в `STDIN` исполняемого файла.

Пример настройки:

```
<source>
<executable>
  <command>cat /opt/dictionaries/os.tsv</command>
  <format>TabSeparated</format>
  <implicit_key>false</implicit_key>
</executable>
</source>
```

Поля настройки:

- `command` — абсолютный путь к исполняемому файлу или имя файла (если каталог программы прописан в `PATH`).
- `format` — формат файла. Поддерживаются все форматы, описанные в разделе [Форматы](#).
- `implicit_key` — исходный исполняемый файл может возвращать только значения, а соответствие запрошенным ключам определено неявно — порядком строк в результате. Значение по умолчанию: `false`. Необязательный параметр.

Этот источник словаря может быть настроен только с помощью XML-конфигурации. Создание словарей с исполняемым источником с помощью DDL запрещено. Иначе пользователь сможет выполнить произвольный бинарный файл на сервере ClickHouse.

## Исполняемый пул

Исполняемый пул позволяет загружать данные из пула процессов. Этот источник не работает со словарями, которые требуют загрузки всех данных из источника. Исполняемый пул работает со словарями, которые размещаются **следующими способами**: `cache`, `complex_key_cache`, `ssd_cache`, `complex_key(ssd)_cache`, `direct`, `complex_key_direct`.

Исполняемый пул генерирует пул процессов с помощью указанной команды и оставляет их активными, пока они не завершатся. Программа считывает данные из потока STDIN пока он доступен и выводит результат в поток STDOUT, а затем ожидает следующего блока данных из STDIN. ClickHouse не закрывает поток STDIN после обработки блока данных и отправляет в него следующую порцию данных, когда это требуется. Исполняемый скрипт должен быть готов к такому способу обработки данных — он должен заранее опрашивать STDIN и отправлять данные в STDOUT.

Пример настройки:

```
<source>
  <executable_pool>
    <command><command>while read key; do printf "$key\tData for key $key\n"; done</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>
```

Поля настройки:

- `command` — абсолютный путь к файлу или имя файла (если каталог программы записан в `PATH`).
- `format` — формат файла. Поддерживаются все форматы, описанные в “[Форматы](#)”.
- `pool_size` — размер пула. Если в поле `pool_size` указан 0, то размер пула не ограничен.
- `command_termination_timeout` — скрипт исполняемого пула должен включать основной цикл чтения-записи. После уничтожения словаря канал закрывается. При этом исполняемый файл имеет `command_termination_timeout` секунд для завершения работы, прежде чем ClickHouse пошлет сигнал SIGTERM дочернему процессу. Указывается в секундах. Значение по умолчанию: 10. Необязательный параметр.
- `max_command_execution_time` — максимальное количество времени для исполняемого скрипта на обработку блока данных. Указывается в секундах. Значение по умолчанию: 10. Необязательный параметр.
- `implicit_key` — исходный исполняемый файл может возвращать только значения, а соответствие запрошенным ключам определено неявно — порядком строк в результате. Значение по умолчанию: `false`. Необязательный параметр.

Этот источник словаря может быть настроен только с помощью XML-конфигурации. Создание словарей с исполняемым источником с помощью DDL запрещено. Иначе пользователь сможет выполнить произвольный бинарный файл на сервере ClickHouse.

## HTTP(s)

Работа с HTTP(s) сервером зависит от [размещения словаря в памяти](#). Если тип размещения словаря `cache` и `complex_key_cache`, то ClickHouse запрашивает необходимые ключи, отправляя запрос методом `POST`.

Пример настройки:

```
<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>
```

или

```
SOURCE(HTTP(
  url 'http://[::1]/os.tsv'
  format 'TabSeparated'
  credentials(user 'user' password 'password')
  headers(header(name 'API-KEY' value 'key'))
))
```

Чтобы ClickHouse смог обратиться к HTTPS-ресурсу, необходимо [настроить openSSL](#) в конфигурации сервера.

Поля настройки:

- `url` — URL источника.
- `format` — формат файла. Поддерживаются все форматы, описанные в разделе [«Форматы»](#).
- `credentials` – базовая HTTP-аутентификация. Необязательный параметр.
- `user` – имя пользователя, необходимое для аутентификации.
- `password` – пароль, необходимый для аутентификации.
- `headers` – все пользовательские записи HTTP-заголовков, используемые для HTTP-запроса. Необязательный параметр.
- `header` – одна запись HTTP-заголовка.
- `name` – идентифицирующее имя, используемое для отправки заголовка запроса.
- `value` – значение, заданное для конкретного идентифицирующего имени.

При создании словаря с помощью DDL-команды (`CREATE DICTIONARY ...`) удаленные хосты для HTTP-словарей проверяются в разделе `remote_url_allow_hosts` из конфигурации сервера. Иначе пользователи базы данных будут иметь доступ к произвольному HTTP-серверу.

## ODBC

Этим способом можно подключить любую базу данных, имеющую ODBC драйвер.

Пример настройки:

```
<source>
  <odbc>
    <db>DatabaseName</db>
    <table>ShemaName.TableName</table>
    <connection_string>DSN=some_parameters</connection_string>
    <invalidate_query>SQL_QUERY</invalidate_query>
  </odbc>
</source>
```

или

```
SOURCE(ODBC(
  db 'DatabaseName'
  table 'SchemaName.TableName'
  connection_string 'DSN=some_parameters'
  invalidate_query 'SQL_QUERY'
))
```

Поля настройки:

- `db` — имя базы данных. Не указывать, если имя базы задано в параметрах `<connection_string>`.
- `table` — имя таблицы и схемы, если она есть.
- `connection_string` — строка соединения.
- `invalidate_query` — запрос для проверки статуса словаря. Необязательный параметр. Читайте подробнее в разделе [Обновление словарей](#).

ClickHouse получает от ODBC-драйвера информацию о квотировании и квотирует настройки в запросах к драйверу, поэтому имя таблицы нужно указывать в соответствии с регистром имени таблицы в базе данных.

Если у вас есть проблемы с кодировками при использовании Oracle, ознакомьтесь с соответствующим разделом [FAQ](#).

## Выявленная уязвимость в функционировании ODBC словарей

### Внимание

При соединении с базой данных через ODBC можно заменить параметр соединения `Servername`. В этом случае, значения `USERNAME` и `PASSWORD` из `odbc.ini` отправляются на удаленный сервер и могут быть скомпрометированы.

### Пример небезопасного использования

Сконфигурируем unixODBC для работы с PostgreSQL. Содержимое `/etc/odbc.ini`:

```
[gregtest]
Driver = /usr/lib/pgsqlodbc.so
Servername = localhost
PORT = 5432
DATABASE = test_db
##OPTION = 3
USERNAME = test
PASSWORD = test
```

Если выполнить запрос вида:

```
SELECT * FROM odbc('DSN=gregtest;Servername=some-server.com', 'test_db');
```

то ODBC драйвер отправит значения `USERNAME` и `PASSWORD` из `odbc.ini` на `some-server.com`.

## Пример подключения PostgreSQL

ОС Ubuntu.

Установка unixODBC и ODBC-драйвера для PostgreSQL:

```
$ sudo apt-get install -y unixodbc odbcinst odbc-postgresql
```

Настройка `/etc/odbc.ini` (или `~/.odbc.ini`):

```
[DEFAULT]
Driver = myconnection

[myconnection]
Description      = PostgreSQL connection to my_db
Driver          = PostgreSQL Unicode
Database        = my_db
Servername      = 127.0.0.1
UserName        = username
Password        = password
Port            = 5432
Protocol        = 9.3
ReadOnly         = No
RowVersioning   = No
ShowSystemTables = No
ConnSettings    =
```

Конфигурация словаря в ClickHouse:

```
<clickhouse>
  <dictionary>
    <name>table_name</name>
    <source>
      <odbc>
        <!-- в connection_string можно указывать следующие параметры: -->
        <!-- 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>
```

или

```
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)
```

Может понадобиться в odbc.ini указать полный путь до библиотеки с драйвером  
DRIVER=/usr/local/lib/psqlodbcw.so.

## Пример подключения MS SQL Server

ОС Ubuntu.

Установка драйвера:

```
$ sudo apt-get install tdsodbc freetds-bin sqsh
```

Настройка драйвера:

```
$ cat /etc/freetds/freetds.conf
...
[MSSQL]
host = 192.168.56.101
port = 1433
tds version = 7.0
client charset = UTF-8

# тестирование TDS соединения
$ 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 # если вы вошли из под пользователя из под которого запущен ClickHouse

[MSSQL]
Description   = FreeTDS
Driver       = FreeTDS
Servername   = MSSQL
Database    = test
UID         = test
PWD         = test
Port        = 1433

# (не обязательно) тест ODBC соединения (используйте isql поставляемый вместе с [unixodbc]
(https://packages.debian.org/sid/unixodbc-package)
$ isql -v MSSQL "user" "password"
```

Примечание:

- чтобы определить самую раннюю версию TDS, которая поддерживается определенной версией  
SQL Server, обратитесь к документации продукта или посмотрите на [MS-TDS Product Behavior](#)

Настройка словаря в 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>
```

ИЛИ

```
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)
```

## СУБД

### MySQL

Пример настройки:

```

<source>
  <mysql>
    <port>3306</port>
    <user>clickhouse</user>
    <password>qwerty</password>
    <replica>
      <host>example01-1</host>
      <priority>1</priority>
    </replica>
    <replica>
      <host>example01-2</host>
      <priority>1</priority>
    </replica>
    <db>db_name</db>
    <table>table_name</table>
    <where>id=10</where>
    <invalidate_query>SQL_QUERY</invalidate_query>
    <fail_on_connection_loss>true</fail_on_connection_loss>
  </mysql>
</source>

```

или

```

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'
))

```

Поля настройки:

- **port** — порт сервера MySQL. Можно указать для всех реплик или для каждой в отдельности (внутри `<replica>`).
- **user** — имя пользователя MySQL. Можно указать для всех реплик или для каждой в отдельности (внутри `<replica>`).
- **password** — пароль пользователя MySQL. Можно указать для всех реплик или для каждой в отдельности (внутри `<replica>`).
- **replica** — блок конфигурации реплики. Блоков может быть несколько.

- `replica/host` — хост MySQL.  
 - `replica/priority` — приоритет реплики. При попытке соединения ClickHouse обходит реплики в соответствии с приоритетом. Чем меньше цифра, тем выше приоритет.

- **db** — имя базы данных.
- **table** — имя таблицы.
- **where** — условие выбора. Синтаксис условия совпадает с синтаксисом секции `WHERE` в MySQL, например, `id > 10 AND id < 20`. Необязательный параметр.
- **invalidate\_query** — запрос для проверки статуса словаря. Необязательный параметр. Читайте подробнее в разделе [Обновление словарей](#).

- `fail_on_connection_loss` – параметр конфигурации, контролирующий поведение сервера при потере соединения. Если значение `true`, то исключение генерируется сразу же, если соединение между клиентом и сервером было потеряно. Если значение `false`, то сервер повторно попытается выполнить запрос три раза прежде чем сгенерировать исключение. Имейте в виду, что повторные попытки могут увеличить время выполнения запроса. Значение по умолчанию: `false`.

MySQL можно подключить на локальном хосте через сокеты, для этого необходимо задать `host` и `socket`.

Пример настройки:

```
<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>
```

ИЛИ

```
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

Пример настройки:

```
<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>
```

ИЛИ

```
SOURCE(CLICKHOUSE(
    host 'example01-01-1'
    port 9000
    user 'default'
    password ""
    db 'default'
    table 'ids'
    where 'id=10'
    secure 1
));
```

Поля настройки:

- `host` — хост ClickHouse. Если `host` локальный, то запрос выполняется без сетевого взаимодействия. Чтобы повысить отказоустойчивость решения, можно создать таблицу типа **Distributed** и прописать её в дальнейших настройках.
- `port` — порт сервера ClickHouse.
- `user` — имя пользователя ClickHouse.
- `password` — пароль пользователя ClickHouse.
- `db` — имя базы данных.
- `table` — имя таблицы.
- `where` — условие выбора. Может отсутствовать.
- `invalidate_query` — запрос для проверки статуса словаря. Необязательный параметр. Читайте подробнее в разделе [Обновление словарей](#).
- `secure` - флаг, разрешающий или не разрешающий защищённое SSL-соединение.

## MongoDB

Пример настройки:

```
<source>
  <mongodb>
    <host>localhost</host>
    <port>27017</port>
    <user></user>
    <password></password>
    <db>test</db>
    <collection>dictionary_source</collection>
  </mongodb>
</source>
```

ИЛИ

```
SOURCE(MONGODB(
    host 'localhost'
    port 27017
    user ""
    password ""
    db 'test'
    collection 'dictionary_source'
))
```

Поля настройки:

- `host` — хост MongoDB.

- `port` — порт сервера MongoDB.
- `user` — имя пользователя MongoDB.
- `password` — пароль пользователя MongoDB.
- `db` — имя базы данных.
- `collection` — имя коллекции.

## Redis

Пример настройки:

```
<source>
  <redis>
    <host>localhost</host>
    <port>6379</port>
    <storage_type>simple</storage_type>
    <db_index>0</db_index>
  </redis>
</source>
```

ИЛИ

```
SOURCE(REDIS
  host 'localhost'
  port 6379
  storage_type 'simple'
  db_index 0
))
```

Поля настройки:

- `host` – хост Redis.
- `port` – порт сервера Redis.
- `storage_type` – способ хранения ключей. Необходимо использовать `simple` для источников с одним столбцом ключей, `hash_map` – для источников с двумя столбцами ключей. Источники с более, чем двумя столбцами ключей, не поддерживаются. Может отсутствовать, значение по умолчанию `simple`.
- `db_index` – номер базы данных. Может отсутствовать, значение по умолчанию 0.

## Cassandra

Пример настройки:

```
<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>
```

Поля настройки:

- `host` – Имя хоста с установленной Cassandra или разделенный через запятую список хостов.
- `port` – Порт на серверах Cassandra. Если не указан, используется значение по умолчанию 9042.
- `user` – Имя пользователя для соединения с Cassandra.
- `password` – Пароль для соединения с Cassandra.
- `keyspace` – Имя keyspace (база данных).
- `column_family` – Имя семейства столбцов (таблица).
- `allow_filtering` – Флаг, разрешающий или не разрешающий потенциально дорогостоящие условия на кластеризации ключевых столбцов. Значение по умолчанию 1.
- `partition_key_prefix` – Количество партиций ключевых столбцов в первичном ключе таблицы Cassandra.

Необходимо для составления ключей словаря. Порядок ключевых столбцов в определении словаря должен быть таким же как в Cassandra.

Значение по умолчанию 1 (первый ключевой столбец это ключ партицирования, остальные ключевые столбцы - ключи кластеризации).

- `consistency` – Уровень консистентности. Возможные значения: `One`, `Two`, `Three`, `All`, `EachQuorum`, `Quorum`, `LocalQuorum`, `LocalOne`, `Serial`, `LocalSerial`. Значение по умолчанию `One`.
- `where` – Опциональный критерий выборки.
- `max_threads` – Максимальное кол-во treadов для загрузки данных из нескольких партиций в словарь.

## PostgreSQL

Пример настройки:

```
<source>
<postgresql>
  <port>5432</port>
  <user>clickhouse</user>
  <password>qwerty</password>
  <db>db_name</db>
  <table>table_name</table>
  <where>id=10</where>
  <invalidate_query>SQL_QUERY</invalidate_query>
</postgresql>
</source>
```

ИЛИ

```
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` – Хост для соединения с PostgreSQL. Вы можете указать его для всех реплик или задать индивидуально для каждой реплики (внутри `<replica>`).
- `port` – Порт для соединения с PostgreSQL. Вы можете указать его для всех реплик или задать индивидуально для каждой реплики (внутри `<replica>`).
- `user` – Имя пользователя для соединения с PostgreSQL. Вы можете указать его для всех реплик или задать индивидуально для каждой реплики (внутри `<replica>`).

- `password` – Пароль для пользователя PostgreSQL.
- `replica` – Раздел конфигурации реплик. Может быть несколько.
  - `replica/host` – хост PostgreSQL.
  - `replica/port` – порт PostgreSQL .
  - `replica/priority` – Приоритет реплики. Во время попытки соединения, ClickHouse будет перебирать реплики в порядке приоритета. Меньшее значение означает более высокий приоритет.
- `db` – Имя базы данных.
- `table` – Имя таблицы.
- `where` – Условие выборки. Синтаксис для условий такой же как для `WHERE` выражения в PostgreSQL, для примера, `id > 10 AND id < 20`. Необязательный параметр.
- `invalidate_query` – Запрос для проверки условия загрузки словаря. Необязательный параметр. Читайте больше в разделе [Обновление словарей](#).

## Ключ и поля словаря

Секция `<structure>` описывает ключ словаря и поля, доступные для запросов.

Описание в формате XML:

```
<dictionary>
  <structure>
    <id>
      <name>Id</name>
    </id>

    <attribute>
      <!-- Attribute parameters -->
    </attribute>

    ...
  </structure>
</dictionary>
```

Атрибуты описываются элементами:

- `<id>` — [столбец с ключом](#).
- `<attribute>` — [столбец данных](#). Можно задать несколько атрибутов.

Создание словаря запросом:

```
CREATE DICTIONARY dict_name (
  Id UInt64,
  -- attributes
)
PRIMARY KEY Id
...
```

Атрибуты задаются в теле запроса:

- `PRIMARY KEY` — [столбец с ключом](#)
- `AttrName AttrType` — [столбец данных](#). Можно задать несколько столбцов.

# Ключ

ClickHouse поддерживает следующие виды ключей:

- Числовой ключ. `UInt64`. Описывается в теге `<id>` или ключевым словом `PRIMARY KEY`.
- Составной ключ. Набор значений разного типа. Описывается в теге `<key>` или ключевым словом `PRIMARY KEY`.

Структура может содержать либо `<id>` либо `<key>`. DDL-запрос может содержать только `PRIMARY KEY`.

## Обратите внимание

Ключ не надо дополнительно описывать в атрибутах.

### Числовой ключ

Тип: `UInt64`.

Пример конфигурации:

```
<id>
  <name>Id</name>
</id>
```

Поля конфигурации:

- `name` — имя столбца с ключами.

Для DDL-запроса:

```
CREATE DICTIONARY (
    Id UInt64,
    ...
)
PRIMARY KEY Id
...
```

- `PRIMARY KEY` — имя столбца с ключами.

### Составной ключ

Ключом может быть кортеж (*tuple*) из полей произвольных типов. В этом случае `layout` должен быть `complex_key_hashed` или `complex_key_cache`.

## Совет

Составной ключ может состоять из одного элемента. Это даёт возможность использовать в качестве ключа, например, строку.

Структура ключа задаётся в элементе `<key>`. Поля ключа задаются в том же формате, что и **атрибуты** словаря. Пример:

```

<structure>
  <key>
    <attribute>
      <name>field1</name>
      <type>String</type>
    </attribute>
    <attribute>
      <name>field2</name>
      <type>UInt32</type>
    </attribute>
  ...
</key>
...

```

или

```

CREATE DICTIONARY (
  field1 String,
  field2 String
  ...
)
PRIMARY KEY field1, field2
...

```

При запросе в функции `dictGet*` в качестве ключа передаётся кортеж. Пример: `dictGetString('dict_name', 'attr_name', tuple('string for field1', num_for_field2))`.

## Атрибуты

Пример конфигурации:

```

<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>

```

или

```

CREATE DICTIONARY somename (
  Name ClickHouseDataType DEFAULT '' EXPRESSION rand64() HIERARCHICAL INJECTIVE IS_OBJECT_ID
)

```

Поля конфигурации:

Тег	Описание	Обязательный
name	Имя столбца.	Да

Тег	Описание	Обязательный
type	<p>Тип данных ClickHouse: <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 пытается привести значение из словаря к заданному типу данных. Например, в случае MySQL, в таблице-источнике поле может быть <code>TEXT</code>, <code>VARCHAR</code>, <code>BLOB</code>, но загружено может быть как <code>String</code>.</p> <p><code>Nullable</code> в настоящее время поддерживается для словарей <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>. Для словарей <code>IPTrie</code> <code>Nullable</code>-типы не поддерживаются.</p>	Да
null_value	<p>Значение по умолчанию для несуществующего элемента.</p> <p>В примере это пустая строка. Значение <code>NULL</code> можно указывать только для типов <code>Nullable</code> (см. предыдущую строку с описанием типов).</p>	Да
expression	<p><b>Выражение</b>, которое ClickHouse выполняет со значением.</p> <p>Выражением может быть имя столбца в удаленной SQL базе.</p> <p>Таким образом, вы можете использовать его для создания псевдонима удаленного столбца.</p> <p>Значение по умолчанию: нет выражения.</p>	Нет
hierarchical	<p>Если <code>true</code>, то атрибут содержит ключ предка для текущего элемента. Смотрите <a href="#">Иерархические словари</a>.</p> <p>Значение по умолчанию: <code>false</code>.</p>	Нет
is_object_id	<p>Признак того, что запрос выполняется к документу MongoDB по <code>ObjectID</code>.</p> <p>Значение по умолчанию: <code>false</code>.</p>	Нет

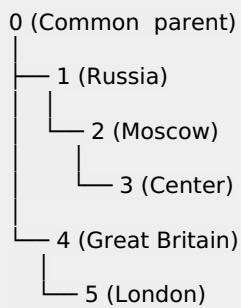
## Смотрите также

- [Функции для работы с внешними словарями](#).

## Иерархические словари

ClickHouse поддерживает иерархические словари с [числовыми ключом](#).

Рассмотрим следующую структуру:



Эту иерархию можно выразить в виде следующей таблицы-словаря.

region_id	parent_region	region_name
1	0	Russia
2	1	Moscow
3	2	Center
4	0	Great Britain
5	4	London

Таблица содержит столбец `parent_region`, содержащий ключ ближайшего предка для текущего элемента.

ClickHouse поддерживает свойство `hierarchical` для атрибутов [внешнего словаря](#). Это свойство позволяет конфигурировать словари, подобные описанному выше.

С помощью функции `dictGetHierarchy` можно получить цепочку предков элемента.

Структура словаря для нашего примера может выглядеть следующим образом:

```

<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>
  
```

## Словари ПОЛИГОНОВ

Словари полигонов позволяют эффективно искать полигон, в который попадают данные точки, среди множества полигонов.

Для примера: определение района города по географическим координатам.

Пример конфигурации:

```
<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>
```

Соответствующий DDL-запрос:

```
CREATE DICTIONARY polygon_dict_name (
  key Array(Array(Array(Array(Float64)))),
  name String,
  value UInt64
)
PRIMARY KEY key
LAYOUT(POLYGON())
...
```

При конфигурации словаря полигонов ключ должен иметь один из двух типов:

- Простой полигон. Представляет из себя массив точек.
- Мультиполигон. Представляет из себя массив полигонов. Каждый полигон задается двумерным массивом точек — первый элемент этого массива задает внешнюю границу полигона, последующие элементы могут задавать дырки, вырезаемые из него.

Точки могут задаваться массивом или кортежем из своих координат. В текущей реализации поддерживается только двумерные точки.

Пользователь может [загружать свои собственные данные](#) во всех поддерживаемых ClickHouse форматах.

Доступно 3 типа хранения данных в памяти:

- POLYGON\_SIMPLE. Это наивная реализация, в которой на каждый запрос делается линейный проход по всем полигонам, и для каждого проверяется принадлежность без использования дополнительных индексов.

- **POLYGON\_INDEX\_EACH**. Для каждого полигона строится отдельный индекс, который позволяет быстро проверять принадлежность в большинстве случаев (оптимизирован под географические регионы).

Также на рассматриваемую область накладывается сетка, которая значительно сужает количество рассматриваемых полигонов.

Сетка строится рекурсивным делением ячейки на 16 равных частей и конфигурируется двумя параметрами.

Деление прекращается при достижении глубины рекурсии **MAX\_DEPTH** или в тот момент, когда ячейку пересекают не более **MIN\_INTERSECTIONS** полигонов.

Для ответа на запрос находится соответствующая ячейка, и происходит поочередное обращение к индексу для сохранных в ней полигонов.

- **POLYGON\_INDEX\_CELL**. В этом размещении также строится сетка, описанная выше. Доступны такие же параметры. Для каждой ячейки-листа строится индекс на всех попадающих в неё кусках полигонов, который позволяет быстро отвечать на запрос.
- **POLYGON**. Синоним к **POLYGON\_INDEX\_CELL**.

Запросы к словарю осуществляются с помощью стандартных [функций](#) для работы со внешними словарями.

Важным отличием является то, что здесь ключами будут являться точки, для которых хочется найти содержащий их полигон.

Пример работы со словарем, определенным выше:

```
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;
```

В результате исполнения последней команды для каждой точки в таблице `points` будет найден полигон минимальной площади, содержащий данную точку, и выведены запрошенные атрибуты.

## Встроенные словари

ClickHouse содержит встроенную возможность работы с геобазой.

Это позволяет:

- для идентификатора региона получить его имя на нужном языке;
- по идентификатору региона получить идентификатор города, области, федерального округа, страны, континента;
- проверить, что один регион входит в другой;
- получить цепочку родительских регионов.

Все функции поддерживают «транслокальность», то есть возможность использовать одновременно разные точки зрения на принадлежность регионов. Подробнее смотрите в разделе «Функции для работы со словарями Яндекс.Метрики».

В пакете по умолчанию, встроенные словари выключены.

Для включения, раскомментируйте параметры `path_to_regions_hierarchy_file` и `path_to_regions_names_files` в конфигурационном файле сервера.

Геобаза загружается из текстовых файлов.

Положите файлы `regions_hierarchy*.txt` в директорию `path_to_regions_hierarchy_file`. Этот конфигурационный параметр должен содержать путь к файлу `regions_hierarchy.txt` (иерархия регионов по умолчанию), а другие файлы (`regions_hierarchy_ua.txt`) должны находиться рядом в той же директории.

Положите файлы `regions_names_*.txt` в директорию `path_to_regions_names_files`.

Также вы можете создать эти файлы самостоятельно. Формат файлов такой:

`regions_hierarchy*.txt`: TabSeparated (без заголовка), столбцы:

- идентификатор региона (`UInt32`);
- идентификатор родительского региона (`UInt32`);
- тип региона (`UInt8`): 1 - континент, 3 - страна, 4 - федеральный округ, 5 - область, 6 - город; остальные типы не имеют значения;
- население (`UInt32`) - не обязательный столбец.

`regions_names_*.txt`: TabSeparated (без заголовка), столбцы:

- идентификатор региона (`UInt32`);
- имя региона (`String`) - не может содержать табы или переводы строк, даже экранированные.

Для хранения в оперативке используется плоский массив. Поэтому, идентификаторы не должны быть больше миллиона.

Словари могут обновляться без перезапуска сервера. Но набор доступных словарей не обновляется. Для обновления проверяется время модификации файлов; если файл изменился, то словарь будет обновлён.

Периодичность проверки настраивается конфигурационным параметром `builtin_dictionaries_reload_interval`.

Обновление словарей (кроме загрузки при первом использовании) не блокирует запросы - во время обновления запросы используют старую версию словарей. Если при обновлении возникнет ошибка, то ошибка пишется в лог сервера, а запросы продолжат использовать старую версию словарей.

Рекомендуется периодически обновлять словари с геобазой. При обновлении, генерируйте новые файлы, записывая их в отдельное место, а только когда всё готово - переименовывайте в файлы, которые использует сервер.

Также имеются функции для работы с идентификаторами операционных систем и поисковых систем Яндекс.Метрики, пользоваться которыми не нужно.

## Типы данных

ClickHouse может сохранять в ячейках таблицы данные различных типов.

Зависимость имен типов данных от регистра можно проверить в системной таблице `system.data_type_families`.

Раздел содержит описания поддерживаемых типов данных и специфику их использования и/или реализации, если таковые имеются.

# UInt8, UInt16, UInt32, UInt64, UInt256, Int8, Int16, Int32, Int64, Int128, Int256

Целые числа фиксированной длины, без знака или со знаком.

При создании таблиц для целых чисел можно указывать числовые параметры (например `TINYINT(8)`, `SMALLINT(16)`, `INT(32)`, `BIGINT(64)`), но ClickHouse их проигнорирует.

## Диапазоны Int

- `Int8` — `[-128 : 127]`
- `Int16` — `[-32768 : 32767]`
- `Int32` — `[-2147483648 : 2147483647]`
- `Int64` — `[-9223372036854775808 : 9223372036854775807]`
- `Int128` — `[-170141183460469231731687303715884105728 : 170141183460469231731687303715884105727]`
- `Int256` — `[-57896044618658097711785492504343953926634992332820282019728792003956564819968 : 57896044618658097711785492504343953926634992332820282019728792003956564819967]`

Синонимы:

- `Int8` — `TINYINT`, `BOOL`, `BOOLEAN`, `INT1`.
- `Int16` — `SMALLINT`, `INT2`.
- `Int32` — `INT`, `INT4`, `INTEGER`.
- `Int64` — `BIGINT`.

## Диапазоны UInt

- `UInt8` — `[0 : 255]`
- `UInt16` — `[0 : 65535]`
- `UInt32` — `[0 : 4294967295]`
- `UInt64` — `[0 : 18446744073709551615]`
- `UInt256` — `[0 : 115792089237316195423570985008687907853269984665640564039457584007913129639935]`

`UInt128` пока не реализован.

## Float32, Float64

Числа с плавающей запятой.

Типы эквивалентны типам языка C:

- `Float32` — `float`.
- `Float64` — `double`.

Синонимы:

- `Float32` — `FLOAT`.
- `Float64` — `DOUBLE`.

При создании таблиц для чисел с плавающей запятой можно указывать числовые параметры (например, `FLOAT(12)`, `FLOAT(15, 22)`, `DOUBLE(12)`, `DOUBLE(4, 18)`), но ClickHouse их проигнорирует.

Рекомендуется хранить данные в целочисленном виде всегда, когда это возможно. Например, переводите в целочисленные значения числа с фиксированной точностью, такие как денежные суммы или времена загрузки страниц в миллисекундах.

## Особенности использования чисел с плавающей запятой

- При вычислениях с числами с плавающей запятой возможна ошибка округления.

```
SELECT 1 - 0.9
```

```
minus(1, 0.9) |  
0.0999999999999998 |
```

- Результат вычисления зависит от метода вычисления (типа процессора и архитектуры вычислительной системы).
- При вычислениях с плавающей запятой возможно появление таких категорий числа как бесконечность (`Inf`) и «не число» (`NaN`). Это необходимо учитывать при обработке результатов вычислений.
- При чтении чисел с плавающей запятой из строк, в качестве результата может быть получено не обязательно ближайшее машинно-представимое число.

## `NaN` и `Inf`

В отличие от стандартного SQL, ClickHouse поддерживает следующие категории чисел с плавающей запятой:

- `Inf` — бесконечность.

```
SELECT 0.5 / 0
```

```
divide(0.5, 0) |  
inf |
```

- `-Inf` — отрицательная бесконечность.

```
SELECT -0.5 / 0
```

```
divide(-0.5, 0) |  
-inf |
```

- `NaN` — не число.

```
SELECT 0 / 0
```

```
divide(0, 0)  
nan |
```

Смотрите правила сортировки `NaN` в разделе [Секция ORDER BY](#).

## Decimal(P, S), Decimal32(S), Decimal64(S), Decimal128(S), Decimal256(S)

Знаковые дробные числа с сохранением точности операций сложения, умножения и вычитания. Для деления осуществляется отбрасывание (не округление) знаков, не попадающих в младший десятичный разряд.

### Параметры

- P - precision. Значение из диапазона [ 1 : 76 ]. Определяет, сколько десятичных знаков (с учетом дробной части) может содержать число.
- S - scale. Значение из диапазона [ 0 : P ]. Определяет, сколько десятичных знаков содержится в дробной части числа.

В зависимости от параметра P `Decimal(P, S)` является синонимом:

- P из [ 1 : 9 ] - для `Decimal32(S)`
- P из [ 10 : 18 ] - для `Decimal64(S)`
- P из [ 19 : 38 ] - для `Decimal128(S)`
- P из [ 39 : 76 ] - для `Decimal256(S)`

### Диапазоны Decimal

- `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)` - (  $-1 * 10^{(76 - S)}$ ,  $1 * 10^{(76 - S)}$  )

Например, `Decimal32(4)` содержит числа от -99999.9999 до 99999.9999 с шагом 0.0001.

### Внутреннее представление

Внутри данные представляются как знаковые целые числа, соответствующей разрядности. Реальные диапазоны, хранящиеся в ячейках памяти несколько больше заявленных. Заявленные диапазоны `Decimal` проверяются только при вводе числа из строкового представления. Поскольку современные CPU не поддерживают 128-битные числа, операции над `Decimal128` эмулируются программно. `Decimal128` работает в разы медленней чем `Decimal32/Decimal64`.

### Операции и типы результата

Результат операции между двумя `Decimal` расширяется до большего типа (независимо от порядка аргументов).

- `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)`

Для размера дробной части (scale) результата действуют следующие правила:

- сложение, вычитание:  $S = \max(S1, S2)$ .
- умножение:  $S = S1 + S2$ .
- деление:  $S = S1$ .

При операциях между `Decimal` и целыми числами результатом является `Decimal`, аналогичный аргументу.

Операции между `Decimal` и `Float32/64` не определены. Для осуществления таких операций нужно явно привести один из аргументов функциями: `toDecimal32`, `toDecimal64`, `toDecimal128`, или `toFloat32`, `toFloat64`. Это сделано из двух соображений. Во-первых, результат операции будет с потерей точности. Во-вторых, преобразование типа - дорогая операция, из-за ее наличия пользовательский запрос может работать в несколько раз дольше.

Часть функций над `Decimal` возвращают `Float64` (например, `var`, `stddev`). Для некоторых из них промежуточные операции проходят в `Decimal`.

Для таких функций результат над одинаковыми данными во `Float64` и `Decimal` может отличаться, несмотря на одинаковый тип результата.

## Проверка переполнений

При выполнении операций над типом `Decimal` могут происходить целочисленные переполнения. Лишняя дробная часть отбрасывается (не округляется). Лишняя целочисленная часть приводит к исключению.

```
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.
```

Проверка переполнения приводит к замедлению операций. При уверенности, что типа результата хватит для его записи проверку переполнения можно отключить настройкой `decimal_check_overflow`. В этом случае при переполнении вернется неверное значение:

```
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

Переполнения происходят не только на арифметических операциях, но и на операциях сравнения. Отключать проверку стоит только при полной уверенности в корректности результата:

```
SELECT toDecimal32(1, 8) < 100
```

DB::Exception: Can't compare.

### Смотрите также

- [isDecimalOverflow](#)
- [countDigits](#)

## Булевые значения

Отдельного типа для булевых значений нет. Для них используется тип UInt8, в котором используются только значения 0 и 1.

## String

Строки произвольной длины. Длина не ограничена. Значение может содержать произвольный набор байт, включая нулевые байты.

Таким образом, тип String заменяет типы VARCHAR, BLOB, CLOB и т. п. из других СУБД.

При создании таблиц для строк можно указывать числовые параметры (например `VARCHAR(255)`), но ClickHouse их проигнорирует.

## Кодировки

В ClickHouse нет понятия кодировок. Строки могут содержать произвольный набор байт, который хранится и выводится, как есть.

Если вам нужно хранить тексты, рекомендуется использовать кодировку UTF-8. По крайней мере, если у вас терминал работает в кодировке UTF-8 (это рекомендуется), вы сможете читать и писать свои значения без каких-либо преобразований.

Также, некоторые функции по работе со строками, имеют отдельные варианты, которые работают при допущении, что строка содержит набор байт, представляющий текст в кодировке UTF-8.

Например, функция `length` вычисляет длину строки в байтах, а функция `lengthUTF8` - длину строки в кодовых точках Unicode, при допущении, что значение в кодировке UTF-8.

## FixedString

Строка фиксированной длины `N` байт (не символов, не кодовых точек).

Чтобы объявить столбец типа `FixedString`, используйте следующий синтаксис:

```
<column_name> FixedString(N)
```

Где N — натуральное число.

Тип FixedString эффективен, когда данные имеют длину ровно N байт. Во всех остальных случаях использование FixedString может привести к снижению эффективности.

Примеры значений, которые можно эффективно хранить в столбцах типа FixedString:

- Двоичное представление IP-адреса (FixedString(16) для IPv6).
- Коды языков (ru\_RU, en\_US ... ).
- Коды валют (USD, RUB ... ).
- Двоичное представление хэшей (FixedString(16) для MD5, FixedString(32) для SHA256).

Для хранения значений UUID используйте тип данных [UUID](#).

При вставке данных, ClickHouse:

- Дополняет строку нулевыми байтами, если строка содержит меньше байтов, чем N.
- Генерирует исключение `Too large value for FixedString(N)`, если строка содержит более N байт.

При выборе данных ClickHouse не обрезает нулевые байты в конце строки. Если вы используете секцию `WHERE`, то необходимо добавлять нулевые байты вручную, чтобы ClickHouse смог сопоставить выражение из фильтра значению `FixedString`. Следующий пример показывает, как использовать секцию `WHERE` с `FixedString`.

Рассмотрим следующую таблицу с единственным столбцом типа `FixedString(2)`:

name
b

Запрос `SELECT * FROM FixedStringTable WHERE a = 'b'` не возвращает необходимых данных. Необходимо дополнить шаблон фильтра нулевыми байтами.

```
SELECT * FROM FixedStringTable  
WHERE a = 'b\0'
```

a
b

Это поведение отличается от поведения MySQL для типа `CHAR`, где строки дополняются пробелами, а пробелы перед выводом вырезаются.

Обратите внимание, что длина значения `FixedString(N)` постоянна. Функция `length` возвращает N даже если значение `FixedString(N)` заполнено только нулевыми байтами, однако функция `empty` в этом же случае возвращает 1.

## UUID

Универсальный уникальный идентификатор (UUID) - это 16-байтовое число, используемое для идентификации записей. Подробнее про UUID читайте на [Википедии](#).

Пример UUID значения представлен ниже:

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

Если при вставке новой записи значение для UUID-колонки не указано, UUID идентификатор будет заполнен нулями:

```
00000000-0000-0000-0000-000000000000
```

## Как сгенерировать UUID

Для генерации UUID-значений предназначена функция [generateUUIDv4](#).

## Примеры использования

Ниже представлены примеры работы с UUID.

### Пример 1

Этот пример демонстрирует, как создать таблицу с UUID-колонкой и добавить в нее сгенерированный UUID.

```
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

### Пример 2

В этом примере, при добавлении записи в таблицу значение для UUID-колонки не задано. UUID будет заполнен нулями.

```
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

## Ограничения

Тип данных UUID можно использовать только с функциями, которые поддерживаются типом данных [String](#) (например, [min](#), [max](#), и [count](#)).

Тип данных UUID не поддерживается арифметическими операциями (например, `abs`) или агрегатными функциями, такими как `sum` и `avg`.

## Date

Дата. Хранится в двух байтах в виде (беззнакового) числа дней, прошедших от 1970-01-01. Позволяет хранить значения от чуть больше, чем начала unix-эпохи до верхнего порога, определяющегося константой на этапе компиляции (сейчас - до 2106 года, последний полностью поддерживаемый год - 2105).

Дата хранится без учёта часового пояса.

### Пример

Создание таблицы и добавление в неё данных:

```
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

### См. также

- [Функции для работы с датой и временем](#)
- [Операторы для работы с датой и временем](#)
- [Тип данных DateTime](#)

## Date32

Дата. Поддерживается такой же диапазон дат, как для типа `Datetime64`. Значение хранится в четырех байтах и соответствует числу дней с 1925-01-01 по 2283-11-11.

### Пример

Создание таблицы со столбцом типа `Date32` и добавление в неё данных:

```
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

## См. также

- [toDate32](#)
- [toDate32OrZero](#)
- [toDate32OrNull](#)

## DateTime

Позволяет хранить момент времени, который может быть представлен как календарная дата и время.

Синтаксис:

```
DateTime([timezone])
```

Диапазон значений: [1970-01-01 00:00:00, 2105-12-31 23:59:59].

Точность: 1 секунда.

## Использование

Момент времени сохраняется как [Unix timestamp](#), независимо от часового пояса и переходов на летнее/зимнее время. Дополнительно, тип `DateTime` позволяет хранить часовой пояс, единый для всей колонки, который влияет на то, как будут отображаться значения типа `DateTime` в текстовом виде и как будут парситься значения заданные в виде строк ('2020-01-01 05:00:01'). Часовой пояс не хранится в строках таблицы (выборки), а хранится в метаданных колонки.

Список поддерживаемых часовых поясов можно найти в [IANA Time Zone Database](#) или получить из базы данных, выполнив запрос `SELECT * FROM system.time_zones`. Также [список](#) есть в Википедии.

Часовой пояс для столбца типа `DateTime` можно в явном виде установить при создании таблицы. Если часовой пояс не установлен, то ClickHouse использует значение параметра `timezone`, установленное в конфигурации сервера или в настройках операционной системы на момент запуска сервера.

Консольный клиент ClickHouse по умолчанию использует часовой пояс сервера, если для значения `DateTime` часовой пояс не был задан в явном виде при инициализации типа данных. Чтобы использовать часовой пояс клиента, запустите `clickhouse-client` с параметром `--use_client_time_zone`.

ClickHouse отображает значения в зависимости от значения параметра `date_time_output_format`. Текстовый формат по умолчанию `YYYY-MM-DD hh:mm:ss`. Кроме того, вы можете поменять отображение с помощью функции `formatDateTime`.

При вставке данных в ClickHouse, можно использовать различные форматы даты и времени в зависимости от значения настройки `date_time_input_format`.

## Примеры

## 1. Создание таблицы с столбцом типа DateTime и вставка данных в неё:

```
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

- При вставке даты-времени как целого числа, оно трактуется как Unix Timestamp (UTC). Unix timestamp `1546300800` в часовом поясе `Europe/London (UTC+0)` представляет время `'2019-01-01 00:00:00'`. Однако, столбец `timestamp` имеет тип `DateTime('Europe/Moscow (UTC+3)')`, так что при выводе в виде строки время отобразится как `2019-01-01 03:00:00`.
- При вставке даты-времени в виде строки, время трактуется соответственно часовому поясу установленному для колонки. `'2019-01-01 00:00:00'` трактуется как время по Москве (и в базу сохраняется `1546290000`)

## 2. Фильтрация по значениям даты-времени

```
SELECT * FROM dt WHERE timestamp = toDateTime('2019-01-01 00:00:00', 'Europe/Moscow')
```

timestamp	event_id
2019-01-01 00:00:00	2

Фильтровать по колонке типа `DateTime` можно, указывая строковое значение в фильтре `WHERE`. Конвертация будет выполнена автоматически:

```
SELECT * FROM dt WHERE timestamp = '2019-01-01 00:00:00'
```

timestamp	event_id
2019-01-01 03:00:00	1

## 3. Получение часового пояса для колонки типа DateTime:

```
SELECT toDateTime(now(), 'Europe/Moscow') AS column, toTypeName(column) AS x
```

column	x
2019-10-16 04:12:04	DateTime('Europe/Moscow')

## 4. Конвертация часовых поясов

```
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

## See Also

- [Функции преобразования типов](#)
- [Функции для работы с датой и временем](#)
- [Функции для работы с массивами](#)
- [Настройка date\\_time\\_input\\_format](#)
- [Настройка date\\_time\\_output\\_format](#)
- [Конфигурационный параметр сервера timezone](#)
- [Операторы для работы с датой и временем](#)
- [Тип данных Date](#)
- [Тип данных DateTime64](#)

## DateTime64

Позволяет хранить момент времени, который может быть представлен как календарная дата и время, с заданной суб-секундной точностью.

Размер тика (точность, precision):  $10^{-precision}$  секунд, где precision - целочисленный параметр.

Возможные значения: [ 0 : 9 ].

Обычно используются - 3 (миллисекунды), 6 (микросекунды), 9 (наносекунды).

### Синтаксис:

```
DateTime64(precision, [timezone])
```

Данные хранятся в виде количества 'тиков', прошедших с момента начала эпохи (1970-01-01 00:00:00 UTC), в Int64. Размер тика определяется параметром precision. Дополнительно, тип DateTime64 позволяет хранить часовой пояс, единый для всей колонки, который влияет на то, как будут отображаться значения типа DateTime64 в текстовом виде и как будут парситься значения заданные в виде строк ('2020-01-01 05:00:01.000'). Часовой пояс не хранится в строках таблицы (выборки), а хранится в метаданных колонки. Подробнее см. [DateTime](#).

Поддерживаются значения от 1 января 1925 г. и до 11 ноября 2283 г.

## Примеры

1. Создание таблицы со столбцом типа DateTime64 и вставка данных в неё:

```
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

- При вставке даты-времени как числа (аналогично ‘Unix timestamp’), время трактуется как UTC. Unix timestamp 1546300800 в часовом поясе Europe/London (UTC+0) представляет время '2019-01-01 00:00:00'. Однако, столбец timestamp имеет тип DateTime('Europe/Moscow (UTC+3)'), так что при выводе в виде строки время отобразится как 2019-01-01 03:00:00.
- При вставке даты-времени в виде строки, время трактуется соответственно часовому поясу установленному для колонки. '2019-01-01 00:00:00' трактуется как время по Москве (и в базу сохраняется '2018-12-31 21:00:00' в виде Unix Timestamp).

## 2. Фильтрация по значениям даты и времени

```
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

В отличие от типа DateTime, DateTime64 не конвертируется из строк автоматически.

## 3. Получение часового пояса для значения типа DateTime64:

```
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. Конвертация часовых поясов

```
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

- [Функции преобразования типов](#)
- [Функции для работы с датой и временем](#)
- [Функции для работы с массивами](#)
- [Настройка date\\_time\\_input\\_format](#)
- [Настройка date\\_time\\_output\\_format](#)
- [Конфигурационный параметр сервера timezone](#)
- [Операторы для работы с датой и временем](#)
- [Тип данных Date](#)
- [Тип данных DateTime](#)

## Enum

Перечисляемый тип данных, содержащий именованные значения.

Именованные значения задаются парами 'string' = integer. ClickHouse хранит только числа, но допускает операции над ними с помощью заданных имён.

ClickHouse поддерживает:

- 8-битный `Enum`. Может содержать до 256 значений, пронумерованных в диапазоне [-128, 127].
- 16-битный `Enum`. Может содержать до 65536 значений, пронумерованных в диапазоне [-32768, 32767].

ClickHouse автоматически выбирает размерность `Enum` при вставке данных. Чтобы точно понимать размер хранимых данных можно использовать типы `Enum8` или `Enum16`.

## Примеры использования

Создадим таблицу со столбцом типа `Enum8('hello' = 1, 'world' = 2)`.

```
CREATE TABLE t_enum
(
    x Enum('hello' = 1, 'world' = 2)
)
ENGINE = TinyLog
```

В столбец `x` можно сохранять только значения, перечисленные при определении типа, т.е. 'hello' или 'world'. Если вы попытаетесь сохранить любое другое значение, ClickHouse сгенерирует исключение. ClickHouse автоматически выберет размерность 8-bit для этого `Enum`.

```
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)

При запросе данных из таблицы ClickHouse выдаст строковые значения из Enum.

```
SELECT * FROM t_enum
```

x
hello
world
hello

Если необходимо увидеть цифровые эквиваленты строкам, то необходимо привести тип Enum к целочисленному.

```
SELECT CAST(x AS Int8) FROM t_enum
```

CAST(x, 'Int8')
1
2
1

Чтобы создать значение типа Enum в запросе, также необходимо использовать функцию CAST.

```
SELECT toTypeName(CAST('a', 'Enum('a' = 1, 'b' = 2)'))
```

```
toTypeName(CAST('a', 'Enum('a' = 1, 'b' = 2)'))—  
Enum8('a' = 1, 'b' = 2) |
```

## Общие правила и особенности использования

Для каждого из значений прописывается число в диапазоне -128 .. 127 для Enum8 или в диапазоне -32768 .. 32767 для Enum16. Все строки должны быть разными, числа - тоже. Разрешена пустая строка. При указании такого типа (в определении таблицы), числа могут идти не подряд и в произвольном порядке. При этом, порядок не имеет значения.

Ни строка, ни цифровое значение в Enum не могут быть **NULL**.

Enum может быть передан в тип **Nullable**. Таким образом, если создать таблицу запросом

```
CREATE TABLE t_enum_nullable  
(  
    x Nullable( Enum8('hello' = 1, 'world' = 2) )  
)  
ENGINE = TinyLog
```

, то в ней можно будет хранить не только 'hello' и 'world', но и NULL.

```
INSERT INTO t_enum_nullable Values('hello'),('world'),(NULL)
```

В оперативке столбец типа Enum представлен так же, как Int8 или Int16 соответствующими числовыми значениями.

При чтении в текстовом виде, парсит значение как строку и ищет соответствующую строку из множества значений Enum-а. Если не находит - кидается исключение.

При записи в текстовом виде, записывает значение как соответствующую строку. Если в данных столбца есть мусор - числа не из допустимого множества, то кидается исключение. При чтении и записи в бинарном виде, оно осуществляется так же, как для типов данных Int8, Int16.

Неявное значение по умолчанию - это значение с минимальным номером.

При ORDER BY, GROUP BY, IN, DISTINCT и т. п., Enum-ы ведут себя так же, как соответствующие числа. Например, при ORDER BY они сортируются по числовым значениям. Функции сравнения на равенство и сравнения на отношение порядка двух Enum-ов работают с Enum-ами так же, как с числами.

Сравнивать Enum с числом нельзя. Можно сравнивать Enum с константной строкой - при этом, для строки ищется соответствующее значение Enum-а; если не находится - кидается исключение.

Поддерживается оператор IN, где слева стоит Enum, а справа - множество строк. В этом случае, строки рассматриваются как значения соответствующего Enum-а.

Большинство операций с числами и со строками не имеет смысла и не работают для Enum-ов: например, к Enum-у нельзя прибавить число.

Для Enum-а естественным образом определяется функция `toString`, которая возвращает его строковое значение.

Также для Enum-а определяются функции `toT`, где T - числовой тип. При совпадении T с типом столбца Enum-а, преобразование работает бесплатно.

При ALTER, есть возможность бесплатно изменить тип Enum-а, если меняется только множество значений. При этом, можно добавлять новые значения; можно удалять старые значения (это безопасно только если они ни разу не использовались, так как это не проверяется). В качестве «защиты от дурака», нельзя менять числовые значения у имеющихся строк - в этом случае, кидается исключение.

При ALTER, есть возможность поменять Enum8 на Enum16 и обратно - так же, как можно поменять Int8 на Int16.

## LowCardinality

Изменяет внутреннее представление других типов данных, превращая их в тип со словарным кодированием.

## Синтаксис

```
LowCardinality(data_type)
```

### Параметры

- `data_type` — `String`, `FixedString`, `Date`, `DateTime` и числа за исключением типа `Decimal`. `LowCardinality` неэффективен для некоторых типов данных, см. описание настройки `allow_suspicious_low_cardinality_types`.

# Описание

`LowCardinality` — это надстройка, изменяющая способ хранения и правила обработки данных. ClickHouse применяет [словарное кодирование](#) в столбцы типа `LowCardinality`. Работа с данными, представленными в словарном виде, может значительно увеличивать производительность запросов `SELECT` для многих приложений.

Эффективность использования типа данных `LowCardinality` зависит от разнообразия данных. Если словарь содержит менее 10 000 различных значений, ClickHouse в основном показывает более высокую эффективность чтения и хранения данных. Если же словарь содержит более 100 000 различных значений, ClickHouse может работать хуже, чем при использовании обычных типов данных.

При работе со строками использование `LowCardinality` вместо `Enum` обеспечивает большую гибкость в использовании и часто показывает такую же или более высокую эффективность.

## Пример

Создание таблицы со столбцами типа `LowCardinality`:

```
CREATE TABLE lc_t
(
    `id` UInt16,
    `strings` LowCardinality(String)
)
ENGINE = MergeTree()
ORDER BY id
```

## Связанные настройки и функции

Настройки:

- [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](#)

Функции:

- [toLowCardinality](#)

## Смотрите также

- [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)

Массив из элементов типа `T`. `T` может любым, в том числе массивом. Таким образом поддерживаются многомерные массивы.

## Создание массива

Массив можно создать с помощью функции:

```
array(T)
```

Также можно использовать квадратные скобки

```
[]
```

Пример создания массива:

```
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)
```

## Особенности работы с типами данных

Максимальный размер массива ограничен одним миллионом элементов.

При создании массива «на лету» ClickHouse автоматически определяет тип аргументов как наиболее узкий тип данных, в котором можно хранить все перечисленные аргументы. Если среди аргументов есть **NULL** или аргумент типа **Nullable**, то тип элементов массива — **Nullable**.

Если ClickHouse не смог подобрать тип данных, то он сгенерирует исключение. Это произойдёт, например, при попытке создать массив одновременно со строками и числами `SELECT array(1, 'a')`.

Примеры автоматического определения типа данных:

```
SELECT array(1, 2, NULL) AS x, toTypeName(x)
```

```
x      toTypeName(array(1, 2, NULL))  
[1,2,NULL] | Array(Nullable(UInt8))
```

Если попытаться создать массив из несовместимых типов данных, то ClickHouse выбросит исключение:

```
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.
```

## Размер массива

Узнать размер массива можно с помощью подстолбца `size0` без чтения всего столбца. Для многомерных массивов можно использовать подстолбец `sizeN-1`, где `N` — требуемое измерение.

## Пример

Запрос:

```
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;
```

Результат:

arr.size0	arr.size1	arr.size2
1   [2]	[[4,1]]	

# AggregateFunction

Агрегатные функции могут обладать определяемым реализацией промежуточным состоянием, которое может быть сериализовано в тип данных, соответствующий `AggregateFunction(...)`, и быть записано в таблицу обычно посредством [материализованного представления] (../../sql-reference/statements/create.md#create-view). Чтобы получить промежуточное состояние, обычно используются агрегатные функции с суффиксом `-State`. Чтобы в дальнейшем получить агрегированные данные необходимо использовать те же агрегатные функции с суффиксом `-Merge`.

`AggregateFunction(name, types_of_arguments...)` — параметрический тип данных.

## Параметры

- Имя агрегатной функции.

Для параметрических агрегатных функций указываются также их параметры.

- Типы аргументов агрегатной функции.

## Пример

```
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`) и `quantiles` — агрегатные функции, поддержанные в ClickHouse.

# Особенности использования

## Вставка данных

Для вставки данных используйте `INSERT SELECT` с агрегатными `-State`-функциями.

## Примеры функций

```
uniqState(UserID)
quantilesState(0.5, 0.9)(SendTiming)
```

В отличие от соответствующих функций `uniq` и `quantiles`, `-State`-функциями возвращают не готовое значение, а состояние. То есть, значение типа `AggregateFunction`.

В запросах `SELECT` значения типа `AggregateFunction` выводятся во всех форматах, которые поддерживает ClickHouse, в виде implementation-specific бинарных данных. Если с помощью `SELECT` выполнить дамп данных, например, в формат `TabSeparated`, то потом этот дамп можно загрузить обратно с помощью запроса `INSERT`.

## Выборка данных

При выборке данных из таблицы `AggregatingMergeTree`, используйте `GROUP BY` и те же агрегатные функции, что и при вставке данных, но с суффиксом `-Merge`.

Агрегатная функция с суффиксом `-Merge` берёт множество состояний, объединяет их, и возвращает результат полной агрегации данных.

Например, следующие два запроса возвращают один и тот же результат:

```
SELECT uniq(UserID) FROM table
SELECT uniqMerge(state) FROM (SELECT uniqState(UserID) AS state FROM table GROUP BY RegionID)
```

## Пример использования

Смотрите в описании движка [AggregatingMergeTree](#).

## Tuple(T1, T2, ...)

Кортеж из элементов любого [типа](#). Элементы кортежа могут быть одного или разных типов.

Кортежи используются для временной группировки столбцов. Столбцы могут группироваться при использовании выражения `IN` в запросе, а также для указания нескольких формальных параметров лямбда-функций. Подробнее смотрите разделы [Операторы IN](#), [Функции высшего порядка](#).

Кортежи могут быть результатом запроса. В этом случае, в текстовых форматах кроме `JSON`, значения выводятся в круглых скобках через запятую. В форматах `JSON`, кортежи выводятся в виде массивов (в квадратных скобках).

## Создание кортежа

Кортеж можно создать с помощью функции

```
tuple(T1, T2, ...)
```

Пример создания кортежа:

```
SELECT tuple(1,'a') AS x, toTypeName(x)
```

```
x-----toTypeName(tuple(1, 'a'))-----
(1,'a') | Tuple(UInt8, String) |
```

# Особенности работы с типами данных

При создании кортежа «на лету» ClickHouse автоматически определяет тип каждого аргументов как минимальный из типов, который может сохранить значение аргумента. Если аргумент — **NULL**, то тип элемента кортежа — **Nullable**.

Пример автоматического определения типа данных:

```
SELECT tuple(1,NULL) AS x, toTypeName(x)
```

```
x-----toTypeName(tuple(1, NULL))-----  
(1,NULL) | Tuple(UInt8, Nullable(Nothing)) |
```

## Адресация элементов кортежа

К элементам кортежа можно обращаться по индексу и по имени:

```
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;
```

Результат:

```
a.s  
y  
x  
tupleElement(a, 2)  
10  
-10
```

# Вложенные структуры данных

## Nested

### Nested(Name1 Type1, Name2 Type2, ...)

Вложенная структура данных - это как будто вложенная таблица. Параметры вложенной структуры данных - имена и типы столбцов, указываются так же, как у запроса CREATE. Каждой строке таблицы может соответствовать произвольное количество строк вложенной структуры данных.

Пример:

```

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)

```

В этом примере объявлена вложенная структура данных `Goals`, содержащая данные о достижении целей. Каждой строке таблицы `visits` может соответствовать от нуля до произвольного количества достижений целей.

Если настройка `flatten_nested` установлена в значение 0 (что не является значением по умолчанию), поддерживаются любые уровни вложенности.

В большинстве случаев, при работе с вложенной структурой данных, указываются отдельные её столбцы. Для этого, имена столбцов указываются через точку. Эти столбцы представляют собой массивы соответствующих типов. Все столбцы-массивы одной вложенной структуры данных имеют одинаковые длины.

Пример:

```

SELECT
    Goals.ID,
    Goals.EventTime
FROM test.visits
WHERE CounterID = 101500 AND length(Goals.ID) < 5
LIMIT 10

```



Проще всего понимать вложенную структуру данных, как набор из нескольких столбцов-массивов одинаковых длин.

Единственное место, где в запросе SELECT можно указать имя целой вложенной структуры данных, а не отдельных столбцов - секция ARRAY JOIN. Подробнее см. раздел «Секция ARRAY JOIN». Пример:

```
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

Вы не можете сделать SELECT целой вложенной структуры данных. Можно лишь явно перечислить отдельные столбцы - её составляющие.

При запросе INSERT, вы должны передать все составляющие столбцы-массивы вложенной структуры данных по отдельности (как если бы это были отдельные столбцы-массивы). При вставке проверяется, что они имеют одинаковые длины.

При запросе DESCRIBE, столбцы вложенной структуры данных перечисляются так же по отдельности.

Работоспособность запроса ALTER для элементов вложенных структур данных, является сильно ограниченной.

## Nullable(TypeName)

Позволяет работать как со значением типа TypeName так и с отсутствием этого значения (NULL) в одной и той же переменной, в том числе хранить NULL в таблицах вместе со значениями типа TypeName. Например, в столбце типа Nullable(Int8) можно хранить значения типа Int8, а в тех строках, где значения нет, будет храниться NULL.

В качестве TypeName нельзя использовать составные типы данных **Array** и **Tuple**. Составные типы данных могут содержать значения типа Nullable, например Array(Nullable(Int8)).

Поле типа Nullable нельзя включать в индексы.

NULL — значение по умолчанию для типа Nullable, если в конфигурации сервера ClickHouse не указано иное.

## Особенности хранения

Для хранения значения типа Nullable ClickHouse использует:

- Отдельный файл с масками NULL (далее маска).
- Непосредственно файл со значениями.

Маска определяет, что лежит в ячейке данных: `NULL` или значение.

В случае, когда маска указывает, что в ячейке хранится `NULL`, в файле значений хранится значение по умолчанию для типа данных. Т.е. если, например, поле имеет тип `Nullable(Int8)`, то ячейка будет хранить значение по умолчанию для `Int8`. Эта особенность увеличивает размер хранилища.

## Info

Почти всегда использование `Nullable` снижает производительность, учитывайте это при проектировании своих баз.

## Поиск `NULL`

Найти в столбце значения `NULL` можно с помощью подстолбца `null`, при этом весь столбец считывать не требуется. Подстолбец содержит `1`, если соответствующее значение равно `NULL`, и `0` если не равно.

### Пример

Запрос:

```
CREATE TABLE nullable (`n` Nullable(UInt32)) ENGINE = MergeTree ORDER BY tuple();
INSERT INTO nullable VALUES (1) (NULL) (2) (NULL);
SELECT n.null FROM nullable;
```

Результат:

n.null
0
1
0
1

## Пример использования

```
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

## Служебные типы данных

Значения служебных типов данных не могут сохраняться в таблицу и выводиться в качестве результата, а возникают как промежуточный результат выполнения запроса.

## Expression

Используется для представления лямбда-выражений в функциях высшего порядка.

## Set

Используется для представления правой части выражения IN.

## Nothing

Этот тип данных предназначен только для того, чтобы представлять **NULL**, т.е. отсутствие значения.

Невозможно создать значение типа Nothing, поэтому он используется там, где значение не подразумевается. Например, **NULL** записывается как **Nullable(Nothing)** (**Nullable** — это тип данных, позволяющий хранить **NULL** в таблицах). Также тип Nothing используется для обозначения пустых массивов:

```
SELECT toTypeName(Array())
```

```
toTypeName(array())  
Array(Nothing)
```

## Interval

Семейство типов данных, представляющих интервалы дат и времени. Оператор **INTERVAL** возвращает значения этих типов.

### Внимание

Нельзя использовать типы данных **Interval** для хранения данных в таблице.

Структура:

- Интервал времени в виде положительного целого числа.
- Тип интервала.

Поддержанные типы интервалов:

- **SECOND**
- **MINUTE**
- **HOUR**
- **DAY**

- WEEK
- MONTH
- QUARTER
- YEAR

Каждому типу интервала соответствует отдельный тип данных. Например, тип данных `IntervalDay` соответствует интервалу `DAY`:

```
SELECT toTypeName(INTERVAL 4 DAY)
```

```
toTypeName(toIntervalDay(4))  
IntervalDay
```

## Использование

Значения типов `Interval` можно использовать в арифметических операциях со значениями типов `Date` и `DateTime`. Например, можно добавить 4 дня к текущей дате:

```
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 |
```

Нельзя объединять интервалы различных типов. Нельзя использовать интервалы вида `4 DAY 1 HOUR`. Вместо этого выражайте интервал в единицах меньших или равных минимальной единице интервала, например, интервал «1 день и 1 час» можно выразить как `25 HOUR` или `90000 SECOND`.

Арифметические операции со значениями типов `Interval` не доступны, однако можно последовательно добавлять различные интервалы к значениям типов `Date` и `DateTime`. Например:

```
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 |
```

Следующий запрос приведёт к генерированию исключения:

```
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..
```

## Смотрите также

- Оператор `INTERVAL`

- Функция приведения типа `toInterval`

## Домены

Домены — это типы данных специального назначения, которые добавляют некоторые дополнительные функции поверх существующего базового типа. На данный момент ClickHouse не поддерживает пользовательские домены.

Вы можете использовать домены везде, где можно использовать соответствующий базовый тип:

- Создание столбца с доменным типом данных.
- Чтение/запись значений из/в столбец с доменным типом данных.
- Используйте его как индекс, если базовый тип можно использовать в качестве индекса.
- Вызов функций со значениями столбца, имеющего доменный тип данных.
- и так далее.

## Дополнительные возможности доменов

- Явное название типа данных столбца в запросах `SHOW CREATE TABLE` и `DESCRIBE TABLE`
- Ввод данных в удобном человеку формате `INSERT INTO domain_table(domain_column) VALUES(...)`
- Вывод данных в удобном человеку формате `SELECT domain_column FROM domain_table`
- Загрузка данных из внешнего источника в удобном для человека формате: `INSERT INTO domain_table FORMAT CSV ...`

## Ограничения

- Невозможно преобразовать базовый тип данных в доменный для индексного столбца с помощью `ALTER TABLE`.
- Невозможно неявно преобразовывать строковые значение в значения с доменным типом данных при вставке данных из другого столбца или таблицы.
- Домен не добавляет ограничения на хранимые значения.

## IPv4

IPv4 — это домен, базирующийся на типе данных `UInt32` предназначенный для хранения адресов IPv4. Он обеспечивает компактное хранение данных с удобным для человека форматом ввода-вывода, и явно отображаемым типом данных в структуре таблицы.

### Применение

```
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				

Или вы можете использовать домен IPv4 в качестве ключа:

```
CREATE TABLE hits (url String, from IPv4) ENGINE = MergeTree() ORDER BY from;
```

IPv4 поддерживает вставку в виде строк с текстовым представлением IPv4 адреса:

```
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

Значения хранятся в компактной бинарной форме:

```
SELECT toTypeName(from), hex(from) FROM hits LIMIT 1;
```

toTypeName(from)	hex(from)
IPv4	B7F7E83A

Значения с доменным типом данных не преобразуются неявно в другие типы данных, кроме UInt32.

Если необходимо преобразовать значение типа IPv4 в строку, то это необходимо делать явно с помощью функции IPv4NumToString():

```
SELECT toTypeName(s), IPv4NumToString(from) AS s FROM hits LIMIT 1;
```

toTypeName(IPv4NumToString(from))	s
String	183.247.232.58

Или приводить к типу данных UInt32:

```
SELECT toTypeName(i), CAST(from AS UInt32) AS i FROM hits LIMIT 1;
```

toTypeName(CAST(from, 'UInt32'))	i
UInt32	3086477370

## IPv6

IPv6 — это домен, базирующийся на типе данных FixedString(16), предназначенный для хранения адресов IPv6. Он обеспечивает компактное хранение данных с удобным для человека форматом ввода-вывода, и явно отображаемым типом данных в структуре таблицы.

### Применение

```
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				

Или вы можете использовать домен IPv6 в качестве ключа:

```
CREATE TABLE hits (url String, from IPv6) ENGINE = MergeTree() ORDER BY from;
```

IPv6 поддерживает вставку в виде строк с текстовым представлением IPv6 адреса:

```
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

Значения хранятся в компактной бинарной форме:

```
SELECT toTypeName(from), hex(from) FROM hits LIMIT 1;
```

toTypeName(from)	hex(from)
IPv6	200144C8012926320033000002520002

Значения с доменным типом данных не преобразуются неявно в другие типы данных, кроме `FixedString(16)`.

Если необходимо преобразовать значение типа `IPv6` в строку, то это необходимо делать явно с помощью функции `IPv6NumToString()`:

```
SELECT toTypeName(s), IPv6NumToString(from) AS s FROM hits LIMIT 1;
```

toTypeName(IPv6NumToString(from))	s
String	2001:44c8:129:2632:33:0:252:2

Или приводить к типу данных `FixedString(16)`:

```
SELECT toTypeName(i), CAST(from AS FixedString(16)) AS i FROM hits LIMIT 1;
```

toTypeName(CAST(from, 'FixedString(16)'))	i
FixedString(16)	◆◆◆

# Составные типы

При создании таблиц вы можете использовать типы данных с названием, состоящим из нескольких слов. Такие названия поддерживаются для лучшей совместимости с SQL.

## Поддержка составных типов

Составные типы	Обычные типы
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
NATIONAL CHARACTER	String
NATIONAL CHAR	String
BINARY LARGE OBJECT	String
BINARY VARYING	String

## Типы данных для работы с географическими структурами

ClickHouse поддерживает типы данных для отображения географических объектов — точек (местоположений), территории и т.п.

### Предупреждение

Сейчас использование типов данных для работы с географическими структурами является экспериментальной возможностью. Чтобы использовать эти типы данных, включите настройку `allow_experimental_geo_types = 1`.

## См. также

- Хранение географических структур данных.
- Настройка `allow_experimental_geo_types`.

## Point

Тип `Point` (точка) определяется парой координат X и Y и хранится в виде кортежа `Tuple(Float64, Float64)`.

### Пример

Запрос:

```
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;
```

Результат:

p	toTypeName(p)
(10,10)	Point

## Ring

Тип `Ring` описывает простой многоугольник без внутренних областей (дыр) и хранится в виде массива точек: `Array(Point)`.

### Пример

Запрос:

```
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;
```

Результат:

r	toTypeName(r)
[(0,0),(10,0),(10,10),(0,10)]	Ring

## Polygon

Тип `Polygon` описывает многоугольник с внутренними областями (дырами) и хранится в виде массива: `Array(Ring)`. Первый элемент массива описывает внешний многоугольник (контур), а остальные элементы описывают дыры.

### Пример

Запись в этой таблице описывает многоугольник с одной дырой:

```
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;
```

Результат:

```
pg [(20,20),(50,20),(50,50),(20,50)],[(30,30),(50,50),(50,30)]] | Polygon      toTypeName(pg)
```

## MultiPolygon

Тип `MultiPolygon` описывает элемент, состоящий из нескольких простых многоугольников (полигональную сетку). Он хранится в виде массива многоугольников: `Array(Polygon)`.

### Пример

Запись в этой таблице описывает элемент, состоящий из двух многоугольников — первый без дыр, а второй с одной дырой:

```
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)

Тип данных `Map(key, value)` хранит пары `ключ:значение`.

### Параметры

- `key` — ключ. `String`, `Integer`, `LowCardinality` или `FixedString`.
- `value` — значение. `String`, `Integer`, `Array`, `LowCardinality` или `FixedString`.

Чтобы получить значение из колонки `a` `Map('key', 'value')`, используйте синтаксис `a['key']`. В настоящее время такая подстановка работает по алгоритму с линейной сложностью.

### Примеры

Рассмотрим таблицу:

```
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);
```

Выборка всех значений ключа `key2`:

```
SELECT a['key2'] FROM table_map;
```

Результат:

```
arrayElement(a, 'key2')  
 10 |  
 20 |  
 30 |
```

Если для какого-то ключа `key` в колонке с типом `Map()` нет значения, запрос возвращает нули для числовых колонок, пустые строки или пустые массивы.

```
INSERT INTO table_map VALUES ( {'key3':100}), ({});  
SELECT a['key3'] FROM table_map;
```

Результат:

```
arrayElement(a, 'key3')  
 100 |  
 0 |  
  
arrayElement(a, 'key3')  
 0 |  
 0 |  
 0 |
```

## Подстолбцы `Map.keys` и `Map.values`

Для оптимизации обработки столбцов `Map` в некоторых случаях можно использовать подстолбцы `keys` и `values` вместо чтения всего столбца.

### Пример

Запрос:

```
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;
```

Результат:

```
a.keys  
['key1','key2','key3'] |  
  
a.values  
[1,2,3] |
```

### См. также

- функция [map\(\)](#)
- функция [CAST\(\)](#)

## SimpleAggregateFunction(func, type)

Хранит только текущее значение агрегатной функции и не сохраняет ее полное состояние, как это делает `AggregateFunction`. Такая оптимизация может быть применена к функциям, которые обладают следующим свойством: результат выполнения функции `f` к набору строк `S1 UNION ALL S2` может быть получен путем выполнения `f` к отдельным частям набора строк, а затем повторного выполнения `f` к результатам:  $f(S1 \text{ UNION ALL } S2) = f(f(S1) \text{ UNION ALL } f(S2))$  Это свойство гарантирует, что результатов частичной агрегации достаточно для вычисления комбинированной, поэтому хранить и обрабатывать какие-либо дополнительные данные не требуется.

Чтобы получить промежуточное значение, обычно используются агрегатные функции с суффиксом `-SimpleState`.

Поддерживаются следующие агрегатные функции:

- `any`
- `anyLast`
- `min`
- `max`
- `sum`
- `sumWithOverflow`
- `groupBitAnd`
- `groupBitOr`
- `groupBitXor`
- `groupArrayArray`
- `groupUniqArrayArray`
- `sumMap`
- `minMap`
- `maxMap`

## Примечание

Значения `SimpleAggregateFunction(func, Type)` отображаются и хранятся так же, как и `Type`, поэтому комбинаторы `-Merge` и `-State` не требуются.

`SimpleAggregateFunction` имеет лучшую производительность, чем `AggregateFunction` с той же агрегатной функцией.

## Параметры

- `func` — имя агрегатной функции.
- `type` — типы аргументов агрегатной функции.

## Пример

```
CREATE TABLE simple (id UInt64, val SimpleAggregateFunction(sum, Double)) ENGINE=AggregatingMergeTree ORDER BY id;
```

## Операторы

Все операторы преобразуются в соответствующие функции на этапе парсинга запроса, с учётом их приоритетов и ассоциативности.

Далее будут перечислены группы операторов в порядке их приоритета (чем выше, тем раньше оператор связывается со своими аргументами).

## Операторы доступа

`a[N]` - доступ к элементу массива, функция `arrayElement(a, N)`.

`a.N` - доступ к элементу кортежа, функция `tupleElement(a, N)`.

## Оператор числового отрицания

`-a` - функция `negate(a)`.

Для чисел в кортеже также может быть использована `tupleNegate`.

## Операторы умножения и деления

`a * b` - функция `multiply(a, b)`

Для умножения кортежа на число также может быть использована `tupleMultiplyByNumber`, для скалярного произведения: `dotProduct`.

`a / b` - функция `divide(a, b)`

Для деления кортежа на число также может быть использована `tupleDivideByNumber`.

`a % b` - функция `modulo(a, b)`

## Операторы сложения и вычитания

`a + b` - функция `plus(a, b)`

Для сложения кортежей также может быть использована `tuplePlus`.

`a - b` - функция `minus(a, b)`

Для вычитания кортежей также может быть использована `tupleMinus`.

## Операторы сравнения

`a = b` - функция `equals(a, b)`

`a == b` - функция `equals(a, b)`

`a != b` - функция `notEquals(a, b)`

`a <> b` - функция `notEquals(a, b)`

`a <= b` - функция `lessOrEquals(a, b)`

`a >= b` - функция `greaterOrEquals(a, b)`

a < b - функция less(a, b)

a > b - функция greater(a, b)

a LIKE s - функция like(a, b)

a NOT LIKE s - функция notLike(a, b)

a ILIKE s – функция ilike(a, b)

a BETWEEN b AND c - равнозначно a >= b AND a <= c

a NOT BETWEEN b AND c - равнозначно a < b OR a > c

## Операторы для работы с множествами

Смотрите раздел [Операторы IN](#).

a IN ... - функция in(a, b)

a NOT IN ... - функция notIn(a, b)

a GLOBAL IN ... - функция globalIn(a, b)

a GLOBAL NOT IN ... - функция globalNotIn(a, b)

a = ANY (subquery) – функция in(a, subquery).

a != ANY (subquery) – равнозначно a NOT IN (SELECT singleValueOrNull(\*) FROM subquery).

a = ALL (subquery) – равнозначно a IN (SELECT singleValueOrNull(\*) FROM subquery).

a != ALL (subquery) – функция notIn(a, subquery).

### Примеры

Запрос с ALL:

```
SELECT number AS a FROM numbers(10) WHERE a > ALL (SELECT number FROM numbers(3, 3));
```

Результат:

a
6
7
8
9

Запрос с ANY:

```
SELECT number AS a FROM numbers(10) WHERE a > ANY (SELECT number FROM numbers(3, 3));
```

Результат:

a
4
5
6
7
8
9

## Оператор для работы с датами и временем

### EXTRACT

```
EXTRACT(part FROM date);
```

Позволяет извлечь отдельные части из переданной даты. Например, можно получить месяц из даты, или минуты из времени.

В параметре `part` указывается, какой фрагмент даты нужно получить. Доступные значения:

- `DAY` — День. Возможные значения: 1-31.
- `MONTH` — Номер месяца. Возможные значения: 1-12.
- `YEAR` — Год.
- `SECOND` — Секунда. Возможные значения: 0-59.
- `MINUTE` — Минута. Возможные значения: 0-59.
- `HOUR` — Час. Возможные значения: 0-23.

Эти значения могут быть указаны также в нижнем регистре (`day`, `month`).

В параметре `date` указывается исходная дата. Поддерживаются типы `Date` и `DateTime`.

Примеры:

```
SELECT EXTRACT(DAY FROM toDate('2017-06-15'));
SELECT EXTRACT(MONTH FROM toDate('2017-06-15'));
SELECT EXTRACT(YEAR FROM toDate('2017-06-15'));
```

В следующем примере создадим таблицу и добавим в неё значение с типом `DateTime`.

```
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

Больше примеров приведено в [тестах](#).

## INTERVAL

Создаёт значение типа **Interval** которое должно использоваться в арифметических операциях со значениями типов **Date** и **DateTime**.

Типы интервалов:

- SECOND
- MINUTE
- HOUR
- DAY
- WEEK
- MONTH
- QUARTER
- YEAR

В качестве значения оператора **INTERVAL** вы можете также использовать строковый литерал.

Например, выражение **INTERVAL 1 HOUR** идентично выражению **INTERVAL '1 hour'** или **INTERVAL '1' hour**.

## Внимание

Интервалы различных типов нельзя объединять. Нельзя использовать выражения вида **INTERVAL 4 DAY 1 HOUR**. Вместо этого интервалы можно выразить в единицах меньших или равных наименьшей единице интервала, Например, **INTERVAL 25 HOUR**. Также можно выполнять последовательные операции как показано в примере ниже.

Примеры:

```
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

Вы можете изменить дату, не используя синтаксис INTERVAL, а просто добавив или отняв секунды, минуты и часы. Например, чтобы передвинуть дату на один день вперед, можно прибавить к ней значение  $60*60*24$ .

## Примечание

Синтаксис INTERVAL или функция addDays предпочтительнее для работы с датами. Сложение с числом (например, синтаксис now() + ...) не учитывает региональные настройки времени, например, переход на летнее время.

Пример:

```
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

## Смотрите также

- Тип данных [Interval](#)
- Функции преобразования типов [toInterval](#)

## Оператор логического "И"

Синтаксис `SELECT a AND b` — вычисляет логическую конъюнкцию между `a` и `b` функцией [and](#).

## Оператор логического "ИЛИ"

Синтаксис `SELECT a OR b` — вычисляет логическую дизъюнкцию между `a` и `b` функцией [or](#).

## Оператор логического отрицания

Синтаксис `SELECT NOT a` — вычисляет логическое отрицание `a` функцией [not](#).

## Условный оператор

`a ? b : c` - функция `if(a, b, c)`

Примечание:

Условный оператор сначала вычисляет значения `b` и `c`, затем проверяет выполнение условия `a`, и только после этого возвращает соответствующее значение. Если в качестве `b` или `c` выступает функция [arrayJoin\(\)](#), то размножение каждой строки произойдет вне зависимости от условия `a`.

## Условное выражение

```
CASE [x]
    WHEN a THEN b
    [WHEN ... THEN ...]
    [ELSE c]
END
```

В случае указания `x` - функция `transform(x, [a, ...], [b, ...], c)`. Иначе — `multilf(a, b, ..., c)`.  
При отсутствии секции `ELSE c`, значением по умолчанию будет `NULL`.

## Примечание

Функция `transform` не умеет работать с `NULL`.

## Оператор склеивания строк

`s1 || s2` - функция `concat(s1, s2)`

## Оператор создания лямбда-выражения

`x -> expr` - функция `lambda(x, expr)`

Следующие операторы не имеют приоритета, так как представляют собой скобки:

## Оператор создания массива

`[x1, ...]` - функция `array(x1, ...)`

## Оператор создания кортежа

`(x1, x2, ...)` - функция `tuple(x2, x2, ...)`

## Ассоциативность

Все бинарные операторы имеют левую ассоциативность. Например, `1 + 2 + 3` преобразуется в `plus(plus(1, 2), 3)`.

Иногда это работает не так, как ожидается. Например, `SELECT 4 > 3 > 2` выдаст 0.

Для эффективности, реализованы функции `and` и `or`, принимающие произвольное количество аргументов. Соответствующие цепочки операторов `AND` и `OR`, преобразуются в один вызов этих функций.

## Проверка на `NULL`

ClickHouse поддерживает операторы `IS NULL` и `IS NOT NULL`.

### IS NULL

- Для значений типа `Nullable` оператор `IS NULL` возвращает:
  - 1, если значение — `NULL`.
  - 0 в обратном случае.
- Для прочих значений оператор `IS NULL` всегда возвращает 0.

Оператор можно оптимизировать, если включить настройку `optimize_functions_to_subcolumns`. При `optimize_functions_to_subcolumns = 1` читается только подстолбец `null` вместо чтения и обработки данных всего столбца. Запрос `SELECT n IS NULL FROM table` преобразуется к запросу `SELECT n.null FROM TABLE`.

```
SELECT x+100 FROM t_null WHERE y IS NULL
```

```
plus(x, 100)  
101 |
```

## IS NOT NULL

- Для значений типа **Nullable** оператор **IS NOT NULL** возвращает:
  - 0, если значение — **NULL**.
  - 1, в обратном случае.
- Для прочих значений оператор **IS NOT NULL** всегда возвращает 1.

Оператор можно оптимизировать, если включить настройку **optimize\_functions\_to\_subcolumns**. При **optimize\_functions\_to\_subcolumns = 1** читается только подстолбец **null** вместо чтения и обработки данных всего столбца. Запрос `SELECT n IS NOT NULL FROM table` преобразуется к запросу `SELECT NOT n.null FROM TABLE`.

```
SELECT * FROM t_null WHERE y IS NOT NULL
```

```
x y  
2 | 3 |
```

## Операторы IN

Операторы **IN**, **NOT IN**, **GLOBAL IN**, **GLOBAL NOT IN** рассматриваются отдельно, так как их функциональность достаточно богатая.

В качестве левой части оператора, может присутствовать как один столбец, так и кортеж.

Примеры:

```
SELECT UserID IN (123, 456) FROM ...  
SELECT (CounterID, UserID) IN ((34, 123), (101500, 456)) FROM ...
```

Если слева стоит один столбец, входящий в индекс, а справа - множество констант, то при выполнении запроса, система воспользуется индексом.

Не перечисляйте слишком большое количество значений (миллионы) явно. Если множество большое - лучше загрузить его во временную таблицу (например, смотрите раздел **Внешние данные для обработки запроса**), и затем воспользоваться подзапросом.

В качестве правой части оператора может быть множество константных выражений, множество кортежей с константными выражениями (показано в примерах выше), а также имя таблицы или подзапрос `SELECT` в скобках.

Если типы данных в левой и правой частях подзапроса **IN** различаются, ClickHouse преобразует значение в левой части к типу данных из правой части. Преобразование выполняется по аналогии с функцией **accurateCastOrNull**, т.е. тип данных становится **Nullable**, а если преобразование не может быть выполнено, возвращается значение **NULL**.

## Пример

Запрос:

```
SELECT '1' IN (SELECT 1);
```

Результат:

in('1', _subquery49)
1

Если в качестве правой части оператора указано имя таблицы (например, UserID IN users), то это эквивалентно подзапросу UserID IN (SELECT \* FROM users). Это используется при работе с внешними данными, отправляемым вместе с запросом. Например, вместе с запросом может быть отправлено множество идентификаторов посетителей, загруженное во временную таблицу users, по которому следует выполнить фильтрацию.

Если в качестве правой части оператора, указано имя таблицы, имеющий движок Set (подготовленное множество, постоянно находящееся в оперативке), то множество не будет создаваться заново при каждом запросе.

В подзапросе может быть указано более одного столбца для фильтрации кортежей.

Пример:

```
SELECT (CounterID, UserID) IN (SELECT CounterID, UserID FROM ...)
```

Типы столбцов слева и справа оператора IN, должны совпадать.

Оператор IN и подзапрос могут встречаться в любой части запроса, в том числе в агрегатных и лямбда функциях.

Пример:

```
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

за каждый день после 17 марта считаем долю хитов, сделанных посетителями, которые заходили на сайт 17 марта.

Подзапрос в секции IN на одном сервере всегда выполняется только один раз. Зависимых подзапросов не существует.

# Обработка NULL

При обработке запроса оператор IN будет считать, что результат операции с `NULL` всегда равен 0, независимо от того, находится `NULL` в правой или левой части оператора. Значения `NULL` не входят ни в какое множество, не соответствуют друг другу и не могут сравниваться, если `transform_null_in = 0`.

Рассмотрим для примера таблицу `t_null`:

x	y
1	NULL
2	3

При выполнении запроса `SELECT x FROM t_null WHERE y IN (NULL,3)` получим следующий результат:

x
2

Видно, что строка, в которой `y = NULL`, выброшена из результатов запроса. Это произошло потому, что ClickHouse не может решить входит ли `NULL` в множество `(NULL,3)`, возвращает результат операции 0, а `SELECT` выбрасывает эту строку из финальной выдачи.

```
SELECT y IN (NULL, 3)
FROM t_null
```

in(y, tuple(NULL, 3))
0
1

## Распределённые подзапросы

Существует два варианта IN-ов с подзапросами (аналогично для JOIN-ов): обычный `IN / JOIN` и `GLOBAL IN / GLOBAL JOIN`. Они отличаются способом выполнения при распределённой обработке запроса.

### Attention

Помните, что алгоритмы, описанные ниже, могут работать иначе в зависимости от [настройки distributed\\_product\\_mode](#).

При использовании обычного IN-а, запрос отправляется на удалённые серверы, и на каждом из них выполняются подзапросы в секциях `IN / JOIN`.

При использовании `GLOBAL IN / GLOBAL JOIN`-а, сначала выполняются все подзапросы для `GLOBAL IN / GLOBAL JOIN`-ов, и результаты складываются во временные таблицы. Затем эти временные таблицы передаются на каждый удалённый сервер, и на них выполняются запросы, с использованием этих переданных временных данных.

Если запрос не распределённый, используйте обычный `IN / JOIN`.

Следует быть внимательным при использовании подзапросов в секции `IN / JOIN` в случае распределённой обработки запроса.

Рассмотрим это на примерах. Пусть на каждом сервере кластера есть обычная таблица **local\_table**. Пусть также есть таблица **distributed\_table** типа **Distributed**, которая смотрит на все серверы кластера.

При запросе к распределённой таблице **distributed\_table**, запрос будет отправлен на все удалённые серверы, и на них будет выполнен с использованием таблицы **local\_table**.

Например, запрос

```
SELECT uniq(UserID) FROM distributed_table
```

будет отправлен на все удалённые серверы в виде

```
SELECT uniq(UserID) FROM local_table
```

, выполнен параллельно на каждом из них до стадии, позволяющей объединить промежуточные результаты; затем промежуточные результаты вернутся на сервер-инициатор запроса, будут на нём объединены, и финальный результат будет отправлен клиенту.

Теперь рассмотрим запрос с IN-ом:

```
SELECT uniq(UserID) FROM distributed_table WHERE CounterID = 101500 AND UserID IN (SELECT UserID FROM local_table WHERE CounterID = 34)
```

- расчёт пересечения аудиторий двух сайтов.

Этот запрос будет отправлен на все удалённые серверы в виде

```
SELECT uniq(UserID) FROM local_table WHERE CounterID = 101500 AND UserID IN (SELECT UserID FROM local_table WHERE CounterID = 34)
```

То есть, множество в секции IN будет собрано на каждом сервере независимо, только по тем данным, которые есть локально на каждом из серверов.

Это будет работать правильно и оптимально, если вы предусмотрели такой случай, и раскладываете данные по серверам кластера таким образом, чтобы данные одного UserID-а лежали только на одном сервере. В таком случае все необходимые данные будут присутствовать на каждом сервере локально. В противном случае результат будет посчитан неточно. Назовём этот вариант запроса «локальный IN».

Чтобы исправить работу запроса, когда данные размазаны по серверам кластера произвольным образом, можно было бы указать **distributed\_table** внутри подзапроса. Запрос будет выглядеть так:

```
SELECT uniq(UserID) FROM distributed_table WHERE CounterID = 101500 AND UserID IN (SELECT UserID FROM distributed_table WHERE CounterID = 34)
```

Этот запрос будет отправлен на все удалённые серверы в виде

```
SELECT uniq(UserID) FROM local_table WHERE CounterID = 101500 AND UserID IN (SELECT UserID FROM distributed_table WHERE CounterID = 34)
```

На каждом удалённом сервере начнёт выполняться подзапрос. Так как в подзапросе используется распределённая таблица, то подзапрос будет, на каждом удалённом сервере, снова отправлен на каждый удалённый сервер, в виде

```
SELECT UserID FROM local_table WHERE CounterID = 34
```

Например, если у вас кластер из 100 серверов, то выполнение всего запроса потребует 10 000 элементарных запросов, что, как правило, является неприемлемым.

В таких случаях всегда следует использовать GLOBAL IN вместо IN. Рассмотрим его работу для запроса

```
SELECT uniq(UserID) FROM distributed_table WHERE CounterID = 101500 AND UserID GLOBAL IN (SELECT UserID FROM distributed_table WHERE CounterID = 34)
```

На сервере-инициаторе запроса будет выполнен подзапрос

```
SELECT UserID FROM distributed_table WHERE CounterID = 34
```

, и результат будет сложен во временную таблицу в оперативке. Затем запрос будет отправлен на каждый удалённый сервер в виде

```
SELECT uniq(UserID) FROM local_table WHERE CounterID = 101500 AND UserID GLOBAL IN _data1
```

, и вместе с запросом, на каждый удалённый сервер будет отправлена временная таблица \_data1 (имя временной таблицы - implementation defined).

Это гораздо более оптимально, чем при использовании обычного IN. Но при этом, следует помнить о нескольких вещах:

1. При создании временной таблицы данные не уникализируются. Чтобы уменьшить объём передаваемых по сети данных, укажите в подзапросе DISTINCT (для обычного IN-а этого делать не нужно).
2. Временная таблица будет передана на все удалённые серверы. Передача не учитывает топологию сети. Например, если 10 удалённых серверов расположены в удалённом относительно сервера-инициатора запроса дата-центре, то по каналу в удалённый дата-центр данные будут переданы 10 раз. Страйтесь не использовать большие множества при использовании GLOBAL IN.
3. При передаче данных на удалённые серверы не настраивается ограничение использования сетевой полосы. Вы можете перегрузить сеть.
4. Страйтесь распределять данные по серверам так, чтобы в GLOBAL IN-ах не было частой необходимости.
5. Если в GLOBAL IN есть частая необходимость, то спланируйте размещение кластера ClickHouse таким образом, чтобы в каждом дата-центре была хотя бы одна реплика каждого шарда, и среди них была быстрая сеть - чтобы запрос целиком можно было бы выполнить, передавая данные в пределах одного дата-центра.

В секции GLOBAL IN также имеет смысл указывать локальную таблицу - в случае, если эта локальная таблица есть только на сервере-инициаторе запроса, и вы хотите воспользоваться данными из неё на удалённых серверах.

## Распределенные подзапросы и max\_parallel\_replicas

Когда настройка max\_parallel\_replicas больше чем 1, распределенные запросы преобразуются. Например, следующий запрос:

```
SELECT CounterID, count() FROM distributed_table_1 WHERE UserID IN (SELECT UserID FROM local_table_2 WHERE CounterID < 100)
SETTINGS max_parallel_replicas=3
```

преобразуются на каждом сервере в

```
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
```

где M значение между 1 и 3 зависящее от того на какой реплике выполняется локальный запрос. Эти параметры влияют на каждую таблицу семейства MergeTree в запросе и имеют тот же эффект, что и применение SAMPLE 1/3 OFFSET (M-1)/3 для каждой таблицы.

Поэтому применение настройки max\_parallel\_replicas даст корректные результаты если обе таблицы имеют одинаковую схему репликации и семплированы по UserID выражению от UserID. В частности, если local\_table\_2 не имеет семплирующего ключа, будут получены неверные результаты. Тоже правило применяется для JOIN.

Один из способов избежать этого, если local\_table\_2 не удовлетворяет требованиям, использовать GLOBAL IN или 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
<b>E011</b>	<b>Numeric data types</b>	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	
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>	Partial	
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	

Feature ID	Feature Name	Status	Comment
<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	
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	

Feature ID	Feature Name	Status	Comment
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>	
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>	

Feature ID	Feature Name	Status	Comment
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	
E091-07	DISTINCT quantifier	Yes	Not all aggregate functions supported
<b>E101</b>	<b>Basic data manipulation</b>	<b>Partial</b>	
E101-01	INSERT statement	Yes	Note: primary key in ClickHouse does not imply the <code>UNIQUE</code> 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>	<b>No</b>	
<b>E121</b>	<b>Basic cursor support</b>	<b>No</b>	
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	

Feature ID	Feature Name	Status	Comment
E131	<b>Null value support (nulls in lieu of values)</b>	Yes	Some restrictions apply
E141	<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	
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	
E151	<b>Transaction support</b>	No	
E151-01	COMMIT statement	No	
E151-02	ROLLBACK statement	No	
E152	<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	
E153	<b>Updatable queries with subqueries</b>	Yes	
E161	<b>SQL comments using leading double minus</b>	Yes	

Feature ID	Feature Name	Status	Comment
E171	<b>SQLSTATE support</b>	No	
E182	<b>Host language binding</b>	No	
F031	<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
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	
F041	<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>Feature ID</b>	<b>Feature Name</b>	<b>Status</b>	<b>Comment</b>
<b>F051</b>	<b>Basic date and time</b>	<b>Partial</b>	
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	
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>	<b>Partial</b>	
<b>F131</b>	<b>Grouped operations</b>	<b>Partial</b>	
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>	<b>No</b>	

Feature ID	Feature Name	Status	Comment
F201	<b>CAST function</b>	Yes	
F221	<b>Explicit defaults</b>	No	
F261	<b>CASE expression</b>	Yes	
F261-01	Simple CASE	Yes	
F261-02	Searched CASE	Yes	
F261-03	NULLIF	Yes	
F261-04	COALESCE	Yes	
F311	<b>Schema definition statement</b>	Partial	
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	
F471	<b>Scalar subquery values</b>	Yes	
F481	<b>Expanded NULL predicate</b>	Yes	
F812	<b>Basic flagging</b>	No	
S011	<b>Distinct data types</b>		
T321	<b>Basic SQL-invoked routines</b>	No	
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	

Feature ID	Feature Name	Status	Comment
T631	<b>IN predicate with one list element</b>	Yes	

## [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> )
<code>WINDOW</code> clause (select ... from table <code>window w as (partiton by id)</code> )	supported
<code>ROWS</code> frame	supported
<code>RANGE</code> frame	supported, the default
<code>INTERVAL</code> syntax for <code>DateTime</code> <code>RANGE OFFSET</code> frame	not supported, specify the number of seconds instead
<code>GROUPS</code> 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>

---

## Руководства

Подробные пошаговые инструкции, которые помогут вам решать различные задачи с помощью ClickHouse.

- [Применение модели CatBoost в ClickHouse](#)
- 

## Применение модели CatBoost в ClickHouse

**CatBoost** — открытая программная библиотека разработанная компанией Яндекс для машинного обучения, которая использует схему градиентного бустинга.

С помощью этой инструкции вы научитесь применять предобученные модели в ClickHouse: в результате вы запустите вывод модели из SQL.

Чтобы применить модель CatBoost в ClickHouse:

1. [Создайте таблицу](#).
2. [Вставьте данные в таблицу](#).
3. [Интегрируйте CatBoost в ClickHouse](#) (Опциональный шаг).
4. [Запустите вывод модели из SQL](#).

Подробнее об обучении моделей в CatBoost, см. [Обучение и применение моделей](#).

Вы можете перегрузить модели CatBoost, если их конфигурация была обновлена, без перезагрузки сервера. Для этого используйте системные запросы [RELOAD MODEL](#) и [RELOAD MODELS](#).

# Перед началом работы

Если у вас еще нет **Docker**, установите его.

## Примечание

**Docker** – это программная платформа для создания контейнеров, которые изолируют установку CatBoost и ClickHouse от остальной части системы.

Перед применением модели CatBoost:

1. Скачайте Docker-образ из реестра:

```
$ docker pull yandex/tutorial-catboost-clickhouse
```

Данный Docker-образ содержит все необходимое для запуска CatBoost и ClickHouse: код, среду выполнения, библиотеки, переменные окружения и файлы конфигурации.

2. Проверьте, что Docker-образ успешно скачался:

```
$ docker image ls
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
yandex/tutorial-catboost-clickhouse  latest    622e4d17945b  22 hours ago  1.37GB
```

3. Запустите Docker-контейнер основанный на данном образе:

```
$ docker run -it -p 8888:8888 yandex/tutorial-catboost-clickhouse
```

## 1. Создайте таблицу

Чтобы создать таблицу для обучающей выборки:

1. Запустите клиент ClickHouse:

```
$ clickhouse client
```

## Примечание

Сервер ClickHouse уже запущен внутри Docker-контейнера.

2. Создайте таблицу в ClickHouse с помощью следующей команды:

```
:) 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. Выйдите из клиента ClickHouse:

```
:) exit
```

## 2. Вставьте данные в таблицу

Чтобы вставить данные:

### 1. Выполните следующую команду:

```
$ clickhouse client --host 127.0.0.1 --query 'INSERT INTO amazon_train FORMAT CSVWithNames' <
~/amazon/train.csv
```

### 2. Запустите клиент ClickHouse:

```
$ clickhouse client
```

### 3. Проверьте, что данные успешно загрузились:

```
:) SELECT count() FROM amazon_train
SELECT count()
FROM amazon_train
+-count()-+
| 65538 |
+-----+
```

## 3. Интегрируйте CatBoost в ClickHouse

### Примечание

**Опциональный шаг.** Docker-образ содержит все необходимое для запуска CatBoost и ClickHouse.

Чтобы интегрировать CatBoost в ClickHouse:

### 1. Создайте библиотеку для оценки модели.

Наиболее быстрый способ оценить модель CatBoost — это скомпилировать библиотеку `libcatboostmodel.<so|dll|dylib>`. Подробнее о том, как скомпилировать библиотеку, читайте в [документации CatBoost](#).

**2.** Создайте в любом месте новую директорию с произвольным названием, например `data` и поместите в нее созданную библиотеку. Docker-образ уже содержит библиотеку `data/libcatboostmodel.so`.

**3.** Создайте в любом месте новую директорию для конфигурации модели с произвольным названием, например `models`.

**4.** Создайте файл конфигурации модели с произвольным названием, например `models/amazon_model.xml`.

**5.** Опишите конфигурацию модели:

```
<models>
  <model>
    <!-- Тип модели. В настоящий момент ClickHouse предоставляет только модель catboost. -->
    <type>catboost</type>
    <!-- Имя модели. -->
    <name>amazon</name>
    <!-- Путь к обученной модели. -->
    <path>/home/catboost/tutorial/catboost_model.bin</path>
    <!-- Интервал обновления. -->
    <lifetime>0</lifetime>
  </model>
</models>
```

**6.** Добавьте в конфигурацию ClickHouse путь к CatBoost и конфигурации модели:

```
<!-- Файл 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>
```

## Примечание

Вы можете позднее изменить путь к конфигурации модели CatBoost без перезагрузки сервера.

## 4. Запустите вывод модели из SQL

Для тестирования модели запустите клиент ClickHouse `$ clickhouse client`.

Проверьте, что модель работает:

```
:) 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
```

## Примечание

Функция **modelEvaluate** возвращает кортежи (tuple) с исходными прогнозами по классам для моделей с несколькими классами.

Спрогнозируйте вероятность:

```
:) 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
```

## Примечание

Подробнее про функцию **exp()**.

Посчитайте логистическую функцию потерь (LogLoss) на всей выборке:

```
:) 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
)
```

## Примечание

Подробнее про функции **avg()**, **log()**.

## Эксплуатация

Руководство по эксплуатации ClickHouse состоит из следующих основных разделов:

- [Требования](#)

- Мониторинг
  - Решение проблем
  - Советы по эксплуатации
  - Процедура обновления
  - Права доступа
  - Резервное копирование
  - Конфигурационные файлы
  - Квоты
  - Системные таблицы
  - Конфигурационные параметры сервера
  - Тестирование серверов с помощью ClickHouse
  - Настройки
  - Утилиты
- 

## Требования

### Процессор

В случае установки из готовых deb-пакетов используйте процессоры с архитектурой x86\_64 и поддержкой инструкций SSE 4.2. Для запуска ClickHouse на процессорах без поддержки SSE 4.2 или на процессорах с архитектурой AArch64 и PowerPC64LE необходимо собирать ClickHouse из исходников.

ClickHouse реализует параллельную обработку данных и использует все доступные аппаратные ресурсы. При выборе процессора учитывайте, что ClickHouse работает более эффективно в конфигурациях с большим количеством ядер, но с более низкой тактовой частотой, чем в конфигурациях с меньшим количеством ядер и более высокой тактовой частотой. Например, 16 ядер с 2600 MHz предпочтительнее, чем 8 ядер с 3600 MHz.

Рекомендуется использовать технологии **Turbo Boost** и **hyper-threading**. Их использование существенно улучшает производительность при типичной нагрузке.

### RAM

Мы рекомендуем использовать как минимум 4 ГБ оперативной памяти, чтобы иметь возможность выполнять нетривиальные запросы. Сервер ClickHouse может работать с гораздо меньшим объёмом RAM, память требуется для обработки запросов.

Необходимый объём RAM зависит от:

- Сложности запросов.
- Объёма данных, обрабатываемых в запросах.

Для расчета объёма RAM необходимо оценить размер промежуточных данных для операций **GROUP BY**, **DISTINCT**, **JOIN** а также других операций, которыми вы пользуетесь.

ClickHouse может использовать внешнюю память для промежуточных данных. Подробнее смотрите в разделе [GROUP BY во внешней памяти](#).

## Файл подкачки

Отключайте файл подкачки в продуктовых средах.

## Подсистема хранения

Для установки ClickHouse необходимо 2ГБ свободного места на диске.

Объём дискового пространства, необходимый для хранения ваших данных, необходимо рассчитывать отдельно. Расчёт должен включать:

- Приблизительную оценку объёма данных.

Можно взять образец данных и получить из него средний размер строки. Затем умножьте полученное значение на количество строк, которое вы планируете хранить.

- Оценку коэффициента сжатия данных.

Чтобы оценить коэффициент сжатия данных, загрузите некоторую выборку данных в ClickHouse и сравните действительный размер данных с размером сохранённой таблицы. Например, данные типа clickstream обычно сжимаются в 6-10 раз.

Для оценки объёма хранилища, примените коэффициент сжатия к размеру данных. Если вы планируете хранить данные в нескольких репликах, то необходимо полученный объём умножить на количество реплик.

## Сеть

По возможности, используйте сети 10G и более высокого класса.

Пропускная способность сети критически важна для обработки распределенных запросов с большим количеством промежуточных данных. Также, скорость сети влияет на задержки в процессах репликации.

## Программное обеспечение

ClickHouse разработан для семейства операционных систем Linux. Рекомендуемый дистрибутив Linux — Ubuntu. В системе должен быть установлен пакет tzdata.

ClickHouse может работать и в других семействах операционных систем. Подробнее смотрите разделе документации [Начало работы](#).

## Мониторинг

Вы можете отслеживать:

- Использование аппаратных ресурсов.
- Метрики сервера ClickHouse.

## Использование ресурсов

ClickHouse не отслеживает состояние аппаратных ресурсов самостоятельно.

Рекомендуем контролировать:

- Загрузку и температуру процессоров.

Можно использовать [dmesg](<https://en.wikipedia.org/wiki/Dmesg>), [turbostat](<https://www.linux.org/docs/man8/turbostat.html>) или другие инструменты.

- Использование системы хранения, оперативной памяти и сети.

## Метрики сервера ClickHouse

Сервер ClickHouse имеет встроенные инструменты мониторинга.

Для отслеживания событий на сервере используйте логи. Подробнее смотрите в разделе конфигурационного файла [logger](#).

ClickHouse собирает:

- Различные метрики того, как сервер использует вычислительные ресурсы.
- Общую статистику обработки запросов.

Метрики находятся в таблицах [system.metrics](#), [system.events](#) и [system.asynchronous\\_metrics](#).

Можно настроить экспорт метрик из ClickHouse в [Graphite](#). Смотрите секцию [graphite](#) конфигурационного файла ClickHouse. Перед настройкой экспорта метрик необходимо настроить Graphite, как указано в [официальном руководстве](#).

Можно настроить экспорт метрик из ClickHouse в [Prometheus](#). Смотрите [prometheus](#) конфигурационного файла ClickHouse. Перед настройкой экспорта метрик необходимо настроить Prometheus, как указано в [официальном руководстве](#).

Также, можно отслеживать доступность сервера через HTTP API. Отправьте HTTP GET к ресурсу /ping. Если сервер доступен, он отвечает 200 OK.

Для мониторинга серверов в кластерной конфигурации необходимо установить параметр [max\\_replica\\_delay\\_for\\_distributed\\_queries](#) и использовать HTTP ресурс /replicas\_status. Если реплика доступна и не отстает от других реплик, то запрос к /replicas\_status возвращает 200 OK. Если реплика отстает, то запрос возвращает 503 HTTP\_SERVICE\_UNAVAILABLE, включая информацию о размере отставания.

## Устранение неисправностей

- [Установка дистрибутива](#)
- [Соединение с сервером](#)
- [Обработка запросов](#)
- [Скорость обработки запросов](#)

### Установка дистрибутива

Не получается скачать deb-пакеты из репозитория ClickHouse с помощью Apt-get

- Проверьте настройки брандмауэра.

- Если по какой-либо причине вы не можете получить доступ к репозиторию, скачайте пакеты как описано в разделе [Начало работы](#) и установите их вручную командой `sudo dpkg -i <packages>`.  
Также, необходим пакет `tzdata`.

## Соединение с сервером

Возможные проблемы:

- Сервер не запущен.
- Неожиданные или неправильные параметры конфигурации.

### Сервер не запущен

#### Проверьте, запущен ли сервер

Команда:

```
$ sudo service clickhouse-server status
```

Если сервер не запущен, запустите его с помощью команды:

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

#### Проверьте журналы

Основной лог `clickhouse-server` по умолчанию — `/var/log/clickhouse-server/clickhouse-server.log`.

В случае успешного запуска вы должны увидеть строки, содержащие:

- `<Information> Application: starting up.` — сервер запускается.
- `<Information> Application: Ready for connections.` — сервер запущен и готов принимать соединения.

Если `clickhouse-server` не запустился из-за ошибки конфигурации вы увидите `<Error>` строку с описанием ошибки. Например:

```
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
```

Если вы не видите ошибки в конце файла, просмотрите весь файл начиная со строки:

```
<Information> Application: starting up.
```

При попытке запустить второй экземпляр `clickhouse-server` журнал выглядит следующим образом:

```
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
```

## Проверьте логи system.d

Если из логов `clickhouse-server` вы не получили необходимой информации или логов нет, то вы можете посмотреть логи `system.d` командой:

```
$ sudo journalctl -u clickhouse-server
```

## Запустите `clickhouse-server` в интерактивном режиме

```
$ sudo -u clickhouse /usr/bin/clickhouse-server --config-file /etc/clickhouse-server/config.xml
```

Эта команда запускает сервер как интерактивное приложение со стандартными параметрами скрипта автозапуска. В этом режиме `clickhouse-server` выводит сообщения в консоль.

## Параметры конфигурации

Проверьте:

- Настройки Docker.

При запуске ClickHouse в Docker в сети IPv6 убедитесь, что установлено `network=host`.

- Параметры endpoint.

Проверьте настройки [listen\_host](#operations-server\_configuration\_parameters-settings-md) и [tcp\_port](#operations-server\_configuration\_parameters-settings-md).

По умолчанию, сервер ClickHouse принимает только локальные подключения.

- Настройки протокола HTTP.

Проверьте настройки протокола для HTTP API.

- Параметры безопасного подключения.

Проверьте:

- Настройку `tcp\_port\_secure`.
- Параметры для SSL-сертификатов.

Используйте правильные параметры при подключении. Например, используйте параметр `port\_secure` при использовании `clickhouse\_client`.

- Настройки пользователей.

Возможно, вы используете неверное имя пользователя или пароль.

## Обработка запросов

Если ClickHouse не может обработать запрос, он отправляет клиенту описание ошибки. В `clickhouse-client` вы получаете описание ошибки в консоли. При использовании интерфейса HTTP, ClickHouse отправляет описание ошибки в теле ответа. Например:

```
$ 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
```

Если вы запускаете `clickhouse-client` с параметром `stack-trace`, то ClickHouse возвращает описание ошибки и соответствующий стек вызовов функций на сервере.

Может появиться сообщение о разрыве соединения. В этом случае необходимо повторить запрос. Если соединение прерывается каждый раз при выполнении запроса, следует проверить журналы сервера на наличие ошибок.

## Скорость обработки запросов

Если вы видите, что ClickHouse работает слишком медленно, необходимо профилировать загрузку ресурсов сервера и сети для ваших запросов.

Для профилирования запросов можно использовать утилиту `clickhouse-benchmark`. Она показывает количество запросов, обработанных за секунду, количество строк, обработанных за секунду и перцентили времени обработки запросов.

## Обновление ClickHouse

Если ClickHouse установлен с помощью deb-пакетов, выполните следующие команды на сервере:

```
$ sudo apt-get update
$ sudo apt-get install clickhouse-client clickhouse-server
$ sudo service clickhouse-server restart
```

Если ClickHouse установлен не из рекомендуемых deb-пакетов, используйте соответствующий метод обновления.

### Примечание

Вы можете обновить сразу несколько серверов, кроме случая, когда все реплики одного шарда отключены.

Обновление ClickHouse до определенной версии:

#### Пример

`xx.yy.a.b` — это номер текущей стабильной версии. Последнюю стабильную версию можно узнать [здесь](#)

```
$ 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
```

## Внешние аутентификаторы пользователей и каталоги

ClickHouse поддерживает аутентификацию и управление пользователями при помощи внешних сервисов.

Поддерживаются следующие внешние аутентификаторы и каталоги:

- [LDAP аутентификатор и каталог](#)
- [Kerberos аутентификатор](#)

## Kerberos

ClickHouse предоставляет возможность аутентификации существующих (и правильно сконфигурированных) пользователей с использованием Kerberos.

В настоящее время возможно использование Kerberos только как внешнего аутентификатора, то есть для аутентификации уже существующих пользователей с помощью Kerberos. Пользователи, настроенные для Kerberos-аутентификации, могут работать с ClickHouse только через HTTP-интерфейс, причём сами клиенты должны иметь возможность аутентификации с использованием механизма GSS-SPNEGO.

!!!

Для Kerberos-аутентификации необходимо предварительно корректно настроить Kerberos на стороне клиента, на сервере и в конфигурационных файлах самого ClickHouse. Ниже описана лишь конфигурация ClickHouse.

## Настройка Kerberos в ClickHouse

Для того, чтобы задействовать Kerberos-аутентификацию в ClickHouse, в первую очередь необходимо добавить одну-единственную секцию `kerberos` в `config.xml`.

В секции могут быть указаны дополнительные параметры:

- `principal` — задаёт имя принципала (canonical service principal name, SPN), используемое при авторизации ClickHouse на Kerberos-сервере.
- Это опциональный параметр, при его отсутствии будет использовано стандартное имя.
- `realm` — обеспечивает фильтрацию по реалм (realm). Пользователям, чей реалм не совпадает с указанным, будет отказано в аутентификации.
- Это опциональный параметр, при его отсутствии фильтр по реалм применяться не будет.

Примеры, как должен выглядеть файл `config.xml`:

```
<clickhouse>
  <!-- ... -->
  <kerberos />
</clickhouse>
```

Или, с указанием принципала:

```
<clickhouse>
  <!-- ... -->
  <kerberos>
    <principal>HTTP/clickhouse.example.com@EXAMPLE.COM</principal>
  </kerberos>
</clickhouse>
```

Или, с фильтрацией по реалм:

```
<clickhouse>
  <!-- ... -->
  <kerberos>
    <realm>EXAMPLE.COM</realm>
  </kerberos>
</clickhouse>
```

## Важно

В конфигурационном файле не могут быть указаны одновременно оба параметра. В противном случае, аутентификация с помощью Kerberos будет недоступна для всех пользователей.

## Важно

В конфигурационном файле может быть не более одной секции `kerberos`. В противном случае, аутентификация с помощью Kerberos будет отключена для всех пользователей.

# Аутентификация пользователей с помощью Kerberos

Уже существующие пользователи могут воспользоваться аутентификацией с помощью Kerberos. Однако, Kerberos-аутентификация возможна только при использовании HTTP-интерфейса.

Имя принципала (principal name) обычно имеет вид:

- `primary/instance@REALM`

Для успешной аутентификации необходимо, чтобы `primary` совпало с именем пользователя ClickHouse, настроенного для использования Kerberos.

## Настройка Kerberos в `users.xml`

Для того, чтобы пользователь имел возможность производить аутентификацию с помощью Kerberos, достаточно включить секцию `kerberos` в описание пользователя в `users.xml` (например, вместо секции `password` или аналогичной ей).

В секции могут быть указаны дополнительные параметры:

- `realm` — обеспечивает фильтрацию по реалм (realm): аутентификация будет возможна только при совпадении реалм клиента с указанным.
- Этот параметр является опциональным, при его отсутствии фильтрация применяться не будет.

Пример, как выглядит конфигурация Kerberos в `users.xml`:

```
<clickhouse>
  <!-- ... -->
  <users>
    <!-- ... -->
    <my_user>
      <!-- ... -->
      <kerberos>
        <realm>EXAMPLE.COM</realm>
      </kerberos>
    </my_user>
  </users>
</clickhouse>
```

## Важно

Если пользователь настроен для Kerberos-аутентификации, другие виды аутентификации будут для него недоступны. Если наряду с `kerberos` в определении пользователя будет указан какой-либо другой способ аутентификации, ClickHouse завершит работу.

Ещё раз отметим, что кроме `users.xml`, необходимо также включить Kerberos в `config.xml`.

## Настройка Kerberos через SQL

Пользователей, использующих Kerberos-аутентификацию, можно создать не только с помощью изменения конфигурационных файлов.

Если SQL-ориентированное управление доступом включено в ClickHouse, можно также создать пользователя, работающего через Kerberos, с помощью SQL.

```
CREATE USER my_user IDENTIFIED WITH kerberos REALM 'EXAMPLE.COM'
```

Или, без фильтрации по реалм:

```
CREATE USER my_user IDENTIFIED WITH kerberos
```

## LDAP

Для аутентификации пользователей ClickHouse можно использовать сервер LDAP. Существуют два подхода:

- Использовать LDAP как внешний аутентификатор для существующих пользователей, которые определены в `users.xml`, или в локальных параметрах управления доступом.
- Использовать LDAP как внешний пользовательский каталог и разрешить аутентификацию локально неопределенных пользователей, если они есть на LDAP сервере.

Для обоих подходов необходимо определить внутреннее имя LDAP сервера в конфигурации ClickHouse, чтобы другие параметры конфигурации могли ссылаться на это имя.

## Определение LDAP сервера

Чтобы определить LDAP сервер, необходимо добавить секцию `ldap_servers` в `config.xml`.

### Пример

```

<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-ECDSSA-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>

```

Обратите внимание, что можно определить несколько LDAP серверов внутри секции `ldap_servers`, используя различные имена.

## Параметры

- `host` — имя хоста сервера LDAP или его IP. Этот параметр обязательный и не может быть пустым.
- `port` — порт сервера LDAP. Если настройка `enable_tls` равна `true`, то по умолчанию используется порт `636`, иначе — порт `389`.
- `bind_dn` — шаблон для создания DN подключения.
  - При формировании DN все подстроки `{user_name}` в шаблоне будут заменяться на фактическое имя пользователя при каждой попытке аутентификации.

- `user_dn_detection` — секция с параметрами LDAP поиска для определения фактического значения DN подключенного пользователя.
  - Это в основном используется в фильтрах поиска для дальнейшего сопоставления ролей, когда сервер является Active Directory. Полученный DN пользователя будет использоваться при замене подстрок `{user_dn}` везде, где они разрешены. По умолчанию DN пользователя устанавливается равным DN подключения, но после выполнения поиска он будет обновлен до фактического найденного значения DN пользователя.
  - `base_dn` — шаблон для создания базового DN для LDAP поиска.
    - При формировании DN все подстроки `{user_name}` и `{bind_dn}` в шаблоне будут заменяться на фактическое имя пользователя и DN подключения соответственно при каждом LDAP поиске.
  - `scope` — область LDAP поиска.
    - Возможные значения: `base`, `one_level`, `children`, `subtree` (по умолчанию).
  - `search_filter` — шаблон для создания фильтра для каждого LDAP поиска.
    - При формировании фильтра все подстроки `{user_name}`, `{bind_dn}`, `{user_dn}` и `{base_dn}` в шаблоне будут заменяться на фактическое имя пользователя, DN подключения, DN пользователя и базовый DN соответственно при каждом LDAP поиске.
  - Обратите внимание, что специальные символы должны быть правильно экранированы в XML.
- `verification_cooldown` — промежуток времени (в секундах) после успешной попытки подключения, в течение которого пользователь будет считаться аутентифицированным и сможет выполнять запросы без повторного обращения к серверам LDAP.
  - Чтобы отключить кеширование и заставить обращаться к серверу LDAP для каждого запроса аутентификации, укажите `0` (значение по умолчанию).
- `enable_tls` — флаг, включающий использование защищенного соединения с сервером LDAP.
  - Укажите `no` для использования текстового протокола `ldap://` (не рекомендовано).
  - Укажите `yes` для обращения к LDAP по протоколу SSL/TLS `ldaps://` (рекомендовано, используется по умолчанию).
  - Укажите `starttls` для использования устаревшего протокола StartTLS (текстовый `ldap://` протокол, модернизированный до TLS).
- `tls_minimum_protocol_version` — минимальная версия протокола SSL/TLS.
  - Возможные значения: `ssl2`, `ssl3`, `tls1.0`, `tls1.1`, `tls1.2` (по-умолчанию).
- `tls_require_cert` — поведение при проверке сертификата SSL/TLS.
  - Возможные значения: `never`, `allow`, `try`, `demand` (по-умолчанию).
- `tls_cert_file` — путь к файлу сертификата.
- `tls_key_file` — путь к файлу ключа сертификата.
- `tls_ca_cert_file` — путь к файлу ЦС (certification authority) сертификата.
- `tls_ca_cert_dir` — путь к каталогу, содержащему сертификаты ЦС.
- `tls_cipher_suite` — разрешенный набор шифров (в нотации OpenSSL).

## Внешний аутентификатор LDAP

Удаленный сервер LDAP можно использовать для верификации паролей локально определенных пользователей (пользователей, которые определены в `users.xml` или в локальных параметрах управления доступом). Для этого укажите имя определенного ранее сервера LDAP вместо `password` или другой аналогичной секции в настройках пользователя.

При каждой попытке авторизации ClickHouse пытается "привязаться" к DN, указанному в [определенении LDAP сервера](#), используя параметр `bind_dn` и предоставленные реквизиты для входа. Если попытка оказалась успешной, пользователь считается аутентифицированным. Обычно это называют методом "простой привязки".

## Пример

```
<clickhouse>
  <!-- ... -->
  <users>
    <!-- ... -->
    <my_user>
      <!-- ... -->
      <ldap>
        <server>my_ldap_server</server>
      </ldap>
    </my_user>
  </users>
</clickhouse>
```

Обратите внимание, что пользователь `my_user` ссылается на `my_ldap_server`. Этот LDAP сервер должен быть настроен в основном файле `config.xml`, как это было описано ранее.

При включенном SQL-ориентированном [управлении доступом](#) пользователи, аутентифицированные LDAP серверами, могут также быть созданы запросом [CREATE USER](#).

Запрос:

```
CREATE USER my_user IDENTIFIED WITH ldap SERVER 'my_ldap_server';
```

## Внешний пользовательский каталог LDAP

В дополнение к локально определенным пользователям, удаленный LDAP сервер может служить источником определения пользователей. Для этого укажите имя определенного ранее сервера LDAP (см. [Определение LDAP сервера](#)) в секции `ldap` внутри секции `users_directories` файла `config.xml`.

При каждой попытке аутентификации ClickHouse пытается локально найти определение пользователя и аутентифицировать его как обычно. Если пользователь не находится локально, ClickHouse предполагает, что он определяется во внешнем LDAP каталоге и пытается "привязаться" к DN, указанному на LDAP сервере, используя предоставленные реквизиты для входа. Если попытка оказалась успешной, пользователь считается существующим и аутентифицированным. Пользователю присваиваются роли из списка, указанного в секции `roles`. Кроме того, если настроена секция `role_mapping`, то выполняется LDAP поиск, а его результаты преобразуются в имена ролей и присваиваются пользователям. Все это работает при условии, что SQL-ориентированное [управлением доступом](#) включено, а роли созданы запросом [CREATE ROLE](#).

## Пример

В `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>

```

Обратите внимание, что `my_ldap_server`, указанный в секции `ldap` внутри секции `user_directories`, должен быть настроен в файле `config.xml`, как это было описано ранее. (см. [Определение LDAP сервера](#)).

## Параметры

- `server` — имя одного из серверов LDAP, определенных в секции `ldap_servers` в файле конфигурации (см. выше). Этот параметр обязательный и не может быть пустым.
- `roles` — секция со списком локально определенных ролей, которые будут присвоены каждому пользователю, полученному от сервера LDAP.
  - Если роли не указаны ни здесь, ни в секции `role_mapping` (см. ниже), пользователь после аутентификации не сможет выполнять никаких действий.

- `role_mapping` — секция с параметрами LDAP поиска и правилами отображения.
  - При аутентификации пользователя, пока еще связанного с LDAP, производится LDAP поиск с помощью `search_filter` и имени этого пользователя. Для каждой записи, найденной в ходе поиска, выделяется значение указанного атрибута. У каждого атрибута, имеющего указанный префикс, этот префикс удаляется, а остальная часть значения становится именем локальной роли, определенной в ClickHouse, причем предполагается, что эта роль была ранее создана запросом `CREATE ROLE` до этого.
- Внутри одной секции `ldap` может быть несколько секций `role_mapping`. Все они будут применены.
  - `base_dn` — шаблон для создания базового DN для LDAP поиска.
    - При формировании DN все подстроки `{user_name}`, `{bind_dn}` и `{user_dn}` в шаблоне будут заменяться на фактическое имя пользователя, DN подключения и DN пользователя соответственно при каждом LDAP поиске.
  - `scope` — область LDAP поиска.
    - Возможные значения: `base`, `one_level`, `children`, `subtree` (по умолчанию).
  - `search_filter` — шаблон для создания фильтра для каждого LDAP поиска.
    - При формировании фильтра все подстроки `{user_name}`, `{bind_dn}`, `{user_dn}` и `{base_dn}` в шаблоне будут заменяться на фактическое имя пользователя, DN подключения, DN пользователя и базовый DN соответственно при каждом LDAP поиске.
  - Обратите внимание, что специальные символы должны быть правильно экранированы в XML.
  - `attribute` — имя атрибута, значение которого будет возвращаться LDAP поиском. По умолчанию: `cn`.
  - `prefix` — префикс, который, как предполагается, будет находиться перед началом каждой строки в исходном списке строк, возвращаемых LDAP поиском. Префикс будет удален из исходных строк, а сами они будут рассматриваться как имена локальных ролей. По умолчанию: пустая строка.

## Управление доступом

ClickHouse поддерживает управление доступом на основе подхода [RBAC](#).

Объекты системы доступа в ClickHouse:

- [Аккаунт пользователя](#)
- [Роль](#)
- [Политика доступа к строкам](#)
- [Профиль настроек](#)
- [Квота](#)

Вы можете настроить объекты системы доступа, используя:

- SQL-ориентированный воркфлоу.
- Функциональность необходимо [включить](#).
- [Конфигурационные файлы](#) сервера: `users.xml` и `config.xml`.

Рекомендуется использовать SQL-воркфлоу. Оба метода конфигурации работают одновременно, поэтому, если для управления доступом вы используете конфигурационные файлы, вы можете плавно перейти на SQL-воркфлоу.

## Внимание

Нельзя одновременно использовать оба метода для управления одним и тем же объектом системы доступа.

Чтобы посмотреть список всех пользователей, ролей, профилей и пр., а также все привилегии, используйте запрос `SHOW ACCESS`.

## Использование

По умолчанию сервер ClickHouse предоставляет аккаунт пользователя `default`, для которого выключена функция SQL-ориентированного управления доступом, но у него есть все права и разрешения. Аккаунт `default` используется во всех случаях, когда имя пользователя не определено. Например, при входе с клиента или в распределенных запросах. При распределенной обработке запроса `default` используется, если в конфигурации сервера или кластера не указаны свойства `user` и `password`.

Если вы начали пользоваться ClickHouse недавно, попробуйте следующий сценарий:

1. Включите SQL-ориентированное управление доступом для пользователя `default`.
2. Войдите под пользователем `default` и создайте всех необходимых пользователей. Не забудьте создать аккаунт администратора (`GRANT ALL ON *.* TO admin_user_account WITH GRANT OPTION`).
3. Ограничьте разрешения для пользователя `default` и отключите для него SQL-ориентированное управление доступом.

## Особенности реализации

- Вы можете выдавать разрешения на базы данных или таблицы, даже если они не существуют.
- При удалении таблицы все связанные с ней привилегии не отзываются. Если вы затем создадите новую таблицу с таким же именем, все привилегии останутся действительными. Чтобы отозвать привилегии, связанные с удаленной таблицей, необходимо выполнить, например, запрос `REVOKE ALL PRIVILEGES ON db.table FROM ALL`.
- У привилегий нет настроек времени жизни.

## Аккаунт пользователя

Аккаунт пользователя — это объект системы доступа, позволяющий авторизовать кого-либо в ClickHouse. Аккаунт содержит:

- Идентификационную информацию.
- **Привилегии**, определяющие область действия запросов, которые могут быть выполнены пользователем.
- Хосты, которые могут подключаться к серверу ClickHouse.
- Назначенные роли и роли по умолчанию.
- Настройки и их ограничения, которые применяются по умолчанию при входе пользователя.
- Присвоенные профили настроек.

Привилегии присваиваются аккаунту пользователя с помощью запроса **GRANT** или через назначение **ролей**. Отозвать привилегию можно с помощью запроса **REVOKE**. Чтобы вывести список присвоенных привилегий, используется выражение **SHOW GRANTS**.

Запросы управления:

- **CREATE USER**
- **ALTER USER**
- **DROP USER**
- **SHOW CREATE USER**

## Применение настроек

Настройки могут быть заданы разными способами: для аккаунта пользователя, для назначенных ему ролей или в профилях настроек. При входе пользователя, если настройка задана для разных объектов системы доступа, значение настройки и ее ограничения применяются в следующем порядке (от высшего приоритета к низшему):

1. Настройки аккаунта.
2. Настройки ролей по умолчанию для аккаунта. Если настройка задана для нескольких ролей, порядок применения не определен.
3. Настройки из профилей настроек, присвоенных пользователю или его ролям по умолчанию. Если настройка задана в нескольких профилях, порядок применения не определен.
4. Настройки, которые по умолчанию применяются ко всему серверу, или настройки из **профиля по умолчанию**.

## Роль

Роль — это контейнер объектов системы доступа, которые можно присвоить аккаунту пользователя.

Роль содержит:

- **Привилегии**
- Настройки и ограничения
- Список назначенных ролей

Запросы управления:

- **CREATE ROLE**
- **ALTER ROLE**
- **DROP ROLE**
- **SET ROLE**
- **SET DEFAULT ROLE**
- **SHOW CREATE ROLE**

Привилегии можно присвоить роли с помощью запроса **GRANT**. Для отзыва привилегий у роли ClickHouse предоставляет запрос **REVOKE**.

## Политика доступа к строкам

Политика доступа к строкам — это фильтр, определяющий, какие строки доступны пользователю или роли. Политика содержит фильтры для конкретной таблицы, а также список ролей и/или пользователей, которые должны использовать данную политику.

Запросы управления:

- [CREATE ROW POLICY](#)
- [ALTER ROW POLICY](#)
- [DROP ROW POLICY](#)
- [SHOW CREATE ROW POLICY](#)

## Профиль настроек

Профиль настроек — это набор [настроек](#). Профиль настроек содержит настройки и ограничения, а также список ролей и/или пользователей, по отношению к которым применяется данный профиль.

Запросы управления:

- [CREATE SETTINGS PROFILE](#)
- [ALTER SETTINGS PROFILE](#)
- [DROP SETTINGS PROFILE](#)
- [SHOW CREATE SETTINGS PROFILE](#)

## Квота

Квота ограничивает использование ресурсов. См. [Квоты](#).

Квота содержит набор ограничений определенной длительности, а также список ролей и/или пользователей, на которых распространяется данная квота.

Запросы управления:

- [CREATE QUOTA](#)
- [ALTER QUOTA](#)
- [DROP QUOTA](#)
- [SHOW CREATE QUOTA](#)

## Включение SQL-ориентированного управления доступом

- Настройте каталог для хранения конфигураций.

ClickHouse хранит конфигурации объектов системы доступа в каталоге, установленном в конфигурационном параметре сервера [access\\_control\\_path](#).

- Включите SQL-ориентированное управление доступом как минимум для одного аккаунта.

По умолчанию управление доступом на основе SQL выключено для всех пользователей. Вам необходимо настроить хотя бы одного пользователя в файле конфигурации `users.xml` и присвоить значение 1 параметру [access\\_management](#).

# Резервное копирование данных

Репликация обеспечивает защиту от аппаратных сбоев, но не защищает от человеческих ошибок: случайного удаления данных, удаления не той таблицы, которую надо было, или таблицы на не том кластере, а также программных ошибок, которые приводят к неправильной обработке данных или их повреждению. Во многих случаях подобные ошибки влияют на все реплики. ClickHouse имеет встроенные средства защиты для предотвращения некоторых типов ошибок — например, по умолчанию **не получится удалить таблицы \*MergeTree, содержащие более 50 Гб данных, одной командой**. Однако эти средства защиты не охватывают все возможные случаи и могут быть обойдены.

Для того чтобы эффективно уменьшить возможные человеческие ошибки, следует тщательно подготовить стратегию резервного копирования и восстановления данных **заранее**.

Каждая компания имеет различные доступные ресурсы и бизнес-требования, поэтому нет универсального решения для резервного копирования и восстановления ClickHouse, которое будет подходить в каждой ситуации. То, что работает для одного гигабайта данных, скорее всего, не будет работать для десятков петабайт. Существует множество возможных подходов со своими плюсами и минусами, которые будут рассмотрены ниже. Рекомендуется использовать несколько подходов вместо одного, чтобы компенсировать их различные недостатки.

## Примечание

Имейте в виду, что если вы создали резервную копию чего-то и никогда не пытались восстановить её, скорее всего, восстановление не будет работать должным образом, когда вам это действительно понадобится (или, по крайней мере, это займет больше времени, чем будет приемлемо для бизнеса). Поэтому, какой бы подход к резервному копированию вы ни выбрали, обязательно автоматизируйте процесс восстановления и регулярно запускайте его на резервном кластере ClickHouse.

## Дублирование данных

Часто данные, которые поступают в ClickHouse, доставляются через некоторую отказоустойчивую очередь, например [Apache Kafka](#). В этом случае можно настроить дополнительный набор подписчиков, которые будут считывать один и тот же поток данных во время записи в ClickHouse и хранить его в холодном хранилище. Большинство компаний уже имеют некоторые рекомендуемые по умолчанию холодные хранилища, которые могут быть хранилищем объектов или распределенной файловой системой, например [HDFS](#).

## Снимки файловой системы

Некоторые локальные файловые системы позволяют делать снимки (например, [ZFS](#)), но они могут быть не лучшим выбором для обслуживания живых запросов. Возможным решением является создание дополнительных реплик с такой файловой системой и исключение их из [Distributed](#) таблиц, используемых для запросов [SELECT](#). Снимки на таких репликах будут недоступны для запросов, изменяющих данные. В качестве бонуса, эти реплики могут иметь особые конфигурации оборудования с большим количеством дисков, подключенных к серверу, что будет экономически эффективным.

## clickhouse-copier

[clickhouse-copier](#) — это универсальный инструмент, который изначально был создан для перешарирования таблиц с петабайтами данных. Его также можно использовать для резервного копирования и восстановления, поскольку он надёжно копирует данные между таблицами и кластерами ClickHouse.

Для небольших объёмов данных можно применять `INSERT INTO ... SELECT ...` в удалённые таблицы.

## Манипуляции сパーティциями

ClickHouse позволяет использовать запрос `ALTER TABLE ... FREEZE PARTITION ...` для создания локальной копииパーティций таблицы. Это реализуется с помощью жестких ссылок (`hardlinks`) на каталог `/var/lib/clickhouse/shadow/`, поэтому такая копия обычно не занимает дополнительное место на диске для старых данных. Созданные копии файлов не обрабатываются сервером ClickHouse, поэтому вы можете просто оставить их там: у вас будет простая резервная копия, которая не требует дополнительной внешней системы, однако при аппаратных проблемах вы можете утратить и актуальные данные и сохраненную копию. По этой причине, лучше удаленно скопировать их в другое место, а затем удалить локальную копию. Распределенные файловые системы и хранилища объектов по-прежнему являются хорошими вариантами для этого, однако можно использовать и обычные присоединенные файловые серверы с достаточно большой ёмкостью (в этом случае передача будет происходить через сетевую файловую систему или, возможно, `rsync`).

Дополнительные сведения о запросах, связанных с манипуляциямиパーティциями, см. в разделе [ALTER](#).

Для автоматизации этого подхода доступен инструмент от сторонних разработчиков: [clickhouse-backup](#).

## Конфигурационные файлы

ClickHouse поддерживает многофайловое управление конфигурацией. Основной конфигурационный файл сервера — `/etc/clickhouse-server/config.xml` или `/etc/clickhouse-server/config.yaml`. Остальные файлы должны находиться в директории `/etc/clickhouse-server/config.d`. Обратите внимание, что конфигурационные файлы могут быть записаны в форматах XML или YAML, но смешение этих форматов в одном файле не поддерживается. Например, можно хранить основные конфигурационные файлы как `config.xml` и `users.xml`, а дополнительные файлы записать в директории `config.d` и `users.d` в формате `.yaml`.

Все XML файлы должны иметь одинаковый корневой элемент, обычно `<clickhouse>`. Для YAML элемент `clickhouse`: должен отсутствовать, так как парсер вставляет его автоматически.

## Переопределение

Некоторые настройки, определенные в основном конфигурационном файле, могут быть переопределены в других файлах:

- У элементов этих конфигурационных файлов могут быть указаны атрибуты `replace` или `remove`.
- Если ни один атрибут не указан, сервер объединит содержимое элементов рекурсивно, заменяя совпадающие значения дочерних элементов.
- Если указан атрибут `replace`, сервер заменит весь элемент на указанный.
- Если указан атрибут `remove`, сервер удалит элемент.

Также возможно указать атрибуты как переменные среды с помощью `from_env="VARIABLE_NAME"`:

```
<clickhouse>
  <macros>
    <replica from_env="REPLICA" />
    <layer from_env="LAYER" />
    <shard from_env="SHARD" />
  </macros>
</clickhouse>
```

## Подстановки

В конфигурационном файле могут быть указаны «подстановки». Если у элемента присутствует атрибут `incl`, то в качестве значения будет использована соответствующая подстановка из файла. По умолчанию путь к файлу с подстановками - `/etc/metrika.xml`. Он может быть изменён в конфигурации сервера в элементе `include_from`. Значения подстановок указываются в элементах `/clickhouse/имя_подстановки` этого файла. Если подстановка, заданная в `incl`, отсутствует, то делается соответствующая запись в лог. Чтобы ClickHouse фиксировал в логе отсутствие подстановки, необходимо указать атрибут `optional="true"` (например, настройки для `macros`).

Если нужно заменить весь элемент подстановкой, можно использовать `include` как имя элемента.

Пример подстановки XML:

```
<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>
```

Подстановки могут также выполняться из ZooKeeper. Для этого укажите у элемента атрибут `from_zk = "/path/to/node"`. Значение элемента заменится на содержимое узла `/path/to/node` в ZooKeeper. В ZooKeeper-узел также можно положить целое XML-поддерево, оно будет целиком вставлено в исходный элемент.

В элементе `users_config` файла `config.xml` можно указать относительный путь к конфигурационному файлу с настройками пользователей, профилей и квот. Значение `users_config` по умолчанию — `users.xml`. Если `users_config` не указан, то настройки пользователей, профилей и квот можно задать непосредственно в `config.xml`.

Настройки пользователя могут быть разделены в несколько отдельных файлов аналогичных `config.xml` и `config.d\`. Имя директории задаётся также как `users_config`.

Имя директории задаётся так же, как имя файла в `users_config`, с подстановкой `.d` вместо `.xml/.yaml`. Директория `users.d` используется по умолчанию, также как `users.xml` используется для `users_config`. Например, можно иметь по отдельному конфигурационному файлу для каждого пользователя:

```
$ 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>
```

Для каждого конфигурационного файла, сервер при запуске генерирует также файлы `file-preprocessed.xml`. Эти файлы содержат все выполненные подстановки и переопределения, и предназначены для информационных целей. Если в конфигурационных файлах были использованы ZooKeeper-подстановки, но при старте сервера ZooKeeper недоступен, то сервер загрузит конфигурацию из `preprocessed`-файла.

Сервер следит за изменениями конфигурационных файлов, а также файлов и ZooKeeper-узлов, которые были использованы при выполнении подстановок и переопределений, и перезагружает настройки пользователей и кластеров на лету. То есть, можно изменять кластера, пользователей и их настройки без перезапуска сервера.

## Примеры записи конфигурации на YAML

Здесь можно рассмотреть пример реальной конфигурации записанной на YAML: [config.yaml.example](#).

Между стандартами XML и YAML имеются различия, поэтому в этом разделе будут перечислены некоторые подсказки для написания конфигурации на YMAL.

Для записи обычной пары ключ-значение следует использовать Scalar:

```
key: value
```

Для создания тега, содержащего подтеги следует использовать Map:

```
map_key:
  key1: val1
  key2: val2
  key3: val3
```

Для создания списка значений или подтегов, расположенных по определенному ключу, следует использовать Sequence:

```
seq_key:
  - val1
  - val2
  - key1: val3
  - map:
    key2: val4
    key3: val5
```

В случае, если необходимо объявить тег, аналогичный XML-атрибуту, необходимо задать скаляр, имеющий ключ с префиксом `@` и заключенный в кавычки:

```
map:  
  "@attr1": value1  
  "@attr2": value2  
  key: 123
```

Из такой Map мы получим после конвертации:

```
<map attr1="value1" attr2="value2">  
  <key>123</key>  
</map>
```

Помимо Map, можно задавать атрибуты для Sequence:

```
seq:  
  - "@attr1": value1  
  - "@attr2": value2  
  - 123  
  - abc
```

Таким образом получая аналог следующей записи на XML:

```
<seq attr1="value1" attr2="value2">123</seq>  
<seq attr1="value1" attr2="value2">abc</seq>
```

## Детали реализации

При старте сервера для каждого конфигурационного файла создаются файлы предобработки `file-preprocessed.xml`. Они содержат все выполненные подстановки и переопределения (эти сведения записываются просто для информации). Если в конфигурационном файле настроены подстановки ZooKeeper, но при старте сервера ZooKeeper не доступен, то сервер загружает конфигурацию из соответствующего файла предобработки.

Сервер отслеживает как изменения в конфигурационных файлах, так и файлы и узы ZooKeeper, которые были использованы при выполнении подстановок и переопределений, и на ходу перезагружает настройки для пользователей и кластеров. Это означает, что можно изменять кластеры, пользователей и их настройки без перезапуска сервера.

## Квоты

Квоты позволяют ограничить использование ресурсов за некоторый интервал времени, или просто подсчитывать использование ресурсов.

Квоты настраиваются в конфиге пользователей. Обычно это `users.xml`.

В системе есть возможность ограничить сложность одного запроса. Для этогосмотрите раздел «Ограничения на сложность запроса».

В отличие от них, квоты:

- ограничивают не один запрос, а множество запросов, которые могут быть выполнены за интервал времени;
- при распределённой обработке запроса, учитывают ресурсы, потраченные на всех удалённых серверах.

Рассмотрим фрагмент файла `users.xml`, описывающего квоты.

```

<!-- Квоты. -->
<quotas>
  <!-- Имя квоты. -->
  <default>
    <!-- Ограничения за интервал времени. Можно задать много интервалов с разными ограничениями. -->
    <interval>
      <!-- Длина интервала. -->
      <duration>3600</duration>

      <!-- Без ограничений. Просто считать соответствующие данные за указанный интервал. -->
      <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>

```

Видно, что квота по умолчанию просто считает использование ресурсов за каждый час, но не ограничивает их.

Подсчитанное использование ресурсов за каждый интервал, выводится в лог сервера после каждого запроса.

```

<statbox>
  <!-- Ограничения за интервал времени. Можно задать много интервалов с разными ограничениями. -->
  <interval>
    <!-- Длина интервала. -->
    <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>

```

Для квоты с именем statbox заданы ограничения за каждый час и за каждые 24 часа (86 400 секунд). Интервал времени считается начиная от некоторого implementation defined фиксированного момента времени. То есть, интервал длины 24 часа начинается не обязательно в полночь.

Когда интервал заканчивается, все накопленные значения сбрасываются. То есть, в следующий час, расчёт квоты за час, начинается заново.

Рассмотрим величины, которые можно ограничить:

`queries` - общее количество запросов;

`query_selects` - общее количество запросов `SELECT`.

`query_inserts` – общее количество запросов `INSERT`.

`errors` – количество запросов, при выполнении которых было выкинуто исключение;

`result_rows` – суммарное количество строк, отанных в виде результата;

`read_rows` – суммарное количество исходных строк, прочитанных из таблиц, для выполнения запроса, на всех удалённых серверах;

`execution_time` – суммарное время выполнения запросов, в секундах (`wall time`);

Если за хотя бы один интервал, ограничение превышено, то кидается исключение с текстом о том, какая величина превышена, за какой интервал, и когда начнётся новый интервал (когда снова можно будет задавать запросы).

Для квоты может быть включена возможность указывать «ключ квоты», чтобы производить учёт ресурсов для многих ключей независимо. Рассмотрим это на примере:

```
<!-- Для глобального конструктора отчётов. -->
<web_global>
  <!-- keyed - значит в параметре запроса передаётся "ключ" quota_key,
       и квота считается по отдельности для каждого значения ключа.
       Например, в качестве ключа может передаваться логин пользователя в Метрике,
       и тогда квота будет считаться для каждого логина по отдельности.
       Имеет смысл использовать только если quota_key передаётся не пользователем, а программой.

  Также можно написать <keyed_by_ip /> - тогда в качестве ключа квоты используется IP-адрес.
  (но стоит учесть, что пользователь может достаточно легко менять IPv6-адрес)
-->
<keyed />
```

Квота прописывается для пользователей в секции `users` конфига. Смотрите раздел «Права доступа».

При распределённой обработке запроса, накопленные величины хранятся на сервере-инициаторе запроса. То есть, если пользователь пойдёт на другой сервер - там квота будет действовать «с нуля».

При перезапуске сервера, квоты сбрасываются.

## 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 doesn't 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 doesn't 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
```

```

/home/milovidov/ClickHouse/build_gcc9/../src/Compression/CompressedReadBufferFromFile.cpp:22
DB::CompressedReadBufferFromFile::nextImpl()
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_data const&,
std::vector<DB::IDataType::Substream, std::allocator<DB::IDataType::Substream> > const&)
/home/milovidov/ClickHouse/build_gcc9/../src/Storages/MergeTree/MergeTreeReader.cpp:212
DB::IDataType::deserializeBinaryBulkWithMultipleStreams(DB::IColumn&, unsigned long,
DB::IDataType::DeserializationSettings&, std::shared_ptr<DB::IDataType::DeserializationState>&) 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
DB::MergeTreeRangeReader::continueReadingChain(DB::MergeTreeRangeReader::ReadResult&)
/home/milovidov/ClickHouse/build_gcc9/../src/Storages/MergeTree/MergeTreeRangeReader.cpp:219
DB::MergeTreeRangeReader::read(unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange> >&)
/home/milovidov/ClickHouse/build_gcc9/../src/Storages/MergeTree/MergeTreeRangeReader.cpp:487
DB::MergeTreeBaseSelectBlockInputStream::readFromPartImpl()
/home/milovidov/ClickHouse/build_gcc9/../src/Storages/MergeTree/MergeTreeBaseSelectBlockInputStream.cpp:158
DB::MergeTreeBaseSelectBlockInputStream::readImpl()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::ExpressionBlockInputStream::readImpl()
/home/milovidov/ClickHouse/build_gcc9/../src/DataStreams/ExpressionBlockInputStream.cpp:34
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::PartialSortingBlockInputStream::readImpl()
/home/milovidov/ClickHouse/build_gcc9/../src/DataStreams/PartialSortingBlockInputStream.cpp:13
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::loop(unsigned long)
/usr/local/include/c++/9.1.0/bits/atomic_base.h:419
DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::thread(std::shared_ptr<DB::ThreadGroupStatus>, unsigned long)
/home/milovidov/ClickHouse/build_gcc9/../src/DataStreams/ParallelInputsProcessor.h:215
ThreadFromGlobalPool::ThreadFromGlobalPool<void
(DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::*)(std::shared_ptr<DB::ThreadGroupStatus>, unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>*, std::shared_ptr<DB::ThreadGroupStatus>, unsigned long&)(void
(DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::*&&)
(std::shared_ptr<DB::ThreadGroupStatus>, unsigned long),
DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>*&&, std::shared_ptr<DB::ThreadGroupStatus>&&, unsigned long&):{lambda()#1}::operator()() const
/usr/local/include/c++/9.1.0/bits/shared_ptr_base.h:729
ThreadPoolImpl<std::thread>::worker(std::list<std::thread>)
/usr/local/include/c++/9.1.0/bits/unique_lock.h:69
execute_native_thread_routine
/home/milovidov/ClickHouse/ci/workspace/gcc/gcc-build/x86_64-pc-linux-gnu/libstdc++-v3/include/bits/unique_ptr.h:81
start_thread

__clone

```

Row 2:

```

count(): 3295
sym: StackTrace::StackTrace(ucontext_t const&)
/home/milovidov/ClickHouse/build_gcc9/../src/Common/StackTrace.cpp:208
DB::(anonymous namespace)::writeTraceInfo(DB::TimerType, int, siginfo_t*, void*) [clone .isra.0]
/home/milovidov/ClickHouse/build_gcc9/../src/IO/BufferBase.h:99

```

\_\_pthread\_cond\_wait

```

std::condition_variable::wait(std::unique_lock<std::mutex>&)
/home/milovidov/ClickHouse/ci/workspace/gcc/gcc-build/x86_64-pc-linux-gnu/libstdc++-
v3/src/c++/11/../../../../gcc-9.1.0/libstdc++-v3/src/c++/11/condition_variable.cc:55
Poco::Semaphore::wait()
/home/milovidov/ClickHouse/build_gcc9/..../contrib/poco/Foundation/src/Semaphore.cpp:61
DB::UnionBlockInputStream::readImpl()
/usr/local/include/c++/9.1.0/x86_64-pc-linux-gnu/bits/gthr-default.h:748
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::MergeSortingBlockInputStream::readImpl()
/home/milovidov/ClickHouse/build_gcc9/..src/Core/Block.h:90
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::ExpressionBlockInputStream::readImpl()
/home/milovidov/ClickHouse/build_gcc9/..src/DataStreams/ExpressionBlockInputStream.cpp:34
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::LimitBlockInputStream::readImpl()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::AsynchronousBlockInputStream::calculate()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
std::function_handler<void (), DB::AsynchronousBlockInputStream::next()>::
{lambda()#1}>::_M_invoke(std::any_data const&)
/usr/local/include/c++/9.1.0/bits/atomic_base.h:551
ThreadPoolImpl<ThreadFromGlobalPool>::worker(std::list<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::MergeTreeRangeSelectBlockInputStream::readFromPartImpl()

```

```

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 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, 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()

```

```
DB::ExpressionBlockInputStream::readImpl()
/home/milovidov/ClickHouse/build_gcc9/../src/DataStreams/ExpressionBlockInputStream.cpp:34
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::PartialSortingBlockInputStream::readImpl()
/home/milovidov/ClickHouse/build_gcc9/../src/DataStreams/PartialSortingBlockInputStream.cpp:13
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::loop(unsigned long)
/usr/local/include/c++/9.1.0/bits/atomic_base.h:419
DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::thread(std::shared_ptr<DB::ThreadGroupStatus>, unsigned long)
/home/milovidov/ClickHouse/build_gcc9/../src/DataStreams/ParallelInputsProcessor.h:215
ThreadFromGlobalPool::ThreadFromGlobalPool<void
(DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::*)(std::shared_ptr<DB::ThreadGroupStatus>, unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>*, std::shared_ptr<DB::ThreadGroupStatus>, unsigned long&)(void
(DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::*&&)
(std::shared_ptr<DB::ThreadGroupStatus>, unsigned long),
DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>*&&, std::shared_ptr<DB::ThreadGroupStatus>&&, unsigned long&){lambda()#1}::operator()() const
/usr/local/include/c++/9.1.0/bits/shared_ptr_base.h:729
ThreadPoolImpl<std::thread>::worker(std::list<std::thread>)
/usr/local/include/c++/9.1.0/bits/unique_lock.h:69
execute_native_thread_routine
/home/milovidov/ClickHouse/ci/workspace/gcc/gcc-build/x86_64-pc-linux-gnu/libstdc++-v3/include/bits/unique_ptr.h:81
start_thread
```

\_clone

Row 5:

```
count(): 1672
sym: StackTrace::StackTrace(ucontext_t const&)
/home/milovidov/ClickHouse/build_gcc9/../src/Common/StackTrace.cpp:208
DB::(anonymous namespace)::writeTraceInfo(DB::TimerType, int, siginfo_t*, void*) [clone .isra.0]
/home/milovidov/ClickHouse/build_gcc9/../src/IO/BufferBase.h:99
```

```
DB::VolnitskyBase<true, true, DB::StringSearcher<true, true> >::search(unsigned char const*, unsigned long) const
/opt/milovidov/ClickHouse/build_gcc9/programs/clickhouse
DB::MatchImpl<true, false>::vector_constant(DB::PODArray<unsigned char, 4096ul, AllocatorWithHint<false, AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul> const&, DB::PODArray<unsigned long, 4096ul, AllocatorWithHint<false, AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul> const&, std::cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&, DB::PODArray<unsigned char, 4096ul, AllocatorWithHint<false, AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul>&)
/opt/milovidov/ClickHouse/build_gcc9/programs/clickhouse
DB::FunctionsStringSearch<DB::MatchImpl<true, false>, DB::NameLike>::executeImpl(DB::Block&, std::vector<unsigned long, std::allocator<unsigned long> > const&, unsigned long, unsigned long)
/opt/milovidov/ClickHouse/build_gcc9/programs/clickhouse
DB::PreparedFunctionImpl::execute(DB::Block&, std::vector<unsigned long, std::allocator<unsigned long> > const&, unsigned long, unsigned long, bool)
/home/milovidov/ClickHouse/build_gcc9/../src/Functions/IFunction.cpp:464
DB::ExpressionAction::execute(DB::Block&, bool) const
/usr/local/include/c++/9.1.0/bits/stl_vector.h:677
DB::ExpressionActions::execute(DB::Block&, bool) const
/home/milovidov/ClickHouse/build_gcc9/../src/Interpreters/ExpressionActions.cpp:739
DB::MergeTreeRangeReader::executePrewhereActionsAndFilterColumns(DB::MergeTreeRangeReader::ReadResult&)
/home/milovidov/ClickHouse/build_gcc9/../src/Storages/MergeTree/MergeTreeRangeReader.cpp:660
DB::MergeTreeRangeReader::read(unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange> >&)
/home/milovidov/ClickHouse/build_gcc9/../src/Storages/MergeTree/MergeTreeRangeReader.cpp:546
DB::MergeTreeRangeReader::read(unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange> >&)
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::MergeTreeBaseSelectBlockInputStream::readFromPartImpl()
/home/milovidov/ClickHouse/build_gcc9/../src/Storages/MergeTree/MergeTreeBaseSelectBlockInputStream.cpp:158
DB::MergeTreeBaseSelectBlockInputStream::readImpl()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::ExpressionBlockInputStream::readImpl()
/home/milovidov/ClickHouse/build_gcc9/../src/DataStreams/ExpressionBlockInputStream.cpp:34
DB::IBlockInputStream::read()
/usr/local/include/c++/9.1.0/bits/stl_vector.h:108
DB::PartialSortingBlockInputStream::readImpl()
/home/milovidov/ClickHouse/build_gcc9/../src/DataStreams/PartialSortingBlockInputStream.cpp:13
DB::IBlockInputStream::read()
```

```

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)

```

```

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::IreadGroupStatus>,  
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
```

\_l11\_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
```

```
IthreadFromGlobalPool::IthreadFromGlobalPool<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<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/MergeTreeRangeReader.cpp:35
DB::MergeTreeRangeReader::startReadingChain(unsigned long, std::vector<DB::MarkRange,
std::allocator<DB::MarkRange> >&)
/home/milovidov/ClickHouse/build_gcc9/..src/Storages/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

```

# Системные таблицы

## Введение

Системные таблицы содержат информацию о:

- состоянии сервера, процессов и окружении.
- внутренних процессах сервера.

Системные таблицы:

- находятся в базе данных `system`.
- доступны только для чтения данных.
- не могут быть удалены или изменены, но их можно отсоединить.

Большинство системных таблиц хранят свои данные в оперативной памяти. Сервер ClickHouse создает эти системные таблицы при старте.

В отличие от других системных таблиц, таблицы с системными логами `metric_log`, `query_log`, `query_thread_log`, `trace_log`, `part_log`, `crash_log` и `text_log` используют движок таблиц `MergeTree` и по умолчанию хранят свои данные в файловой системе. Если удалить таблицу из файловой системы, сервер ClickHouse снова создаст пустую таблицу во время следующей записи данных. Если схема системной таблицы изменилась в новом релизе, то ClickHouse переименует текущую таблицу и создаст новую.

Таблицы с системными логами `log` можно настроить, создав конфигурационный файл с тем же именем, что и таблица в разделе `/etc/clickhouse-server/config.d/`, или указав соответствующие элементы в `/etc/clickhouse-server/config.xml`. Настраиваться могут следующие элементы:

- `database` — база данных, к которой принадлежит системная таблица. Эта опция на текущий момент устарела. Все системные таблицы находятся в базе данных `system`.
- `table` — таблица для добавления данных.
- `partition_by` — **ключ партиционирования**.
- `ttl` — **время жизни** записей в таблице.
- `flush_interval_milliseconds` — интервал сброса данных на диск, в миллисекундах.
- `engine` — полное имя движка (начиная с `ENGINE =` ) с параметрами. Эта опция противоречит `partition_by` и `ttl`. Если указать оба параметра вместе, сервер вернет ошибку и завершит работу.

Пример:

```
<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>
```

По умолчанию размер таблицы не ограничен. Управлять размером таблицы можно используя [TTL](#) для удаления устаревших записей журнала. Также вы можете использовать функциюパーティционирования для таблиц `MergeTree`.

## Источники системных показателей

Для сбора системных показателей сервер ClickHouse использует:

- возможности `CAP_NET_ADMIN`.
- `procfs` (только Linux).

Если для сервера ClickHouse не включено `CAP_NET_ADMIN`, он пытается обратиться к `ProcfsMetricsProvider`. `ProcfsMetricsProvider` позволяет собирать системные показатели для каждого запроса (для CPU и I/O).

Если `procfs` поддерживается и включена в системе, то сервер ClickHouse собирает следующие системные показатели:

- `OSCPUVirtualTimeMicroseconds`
- `OSCPUWaitMicroseconds`
- `OSIOWaitMicroseconds`
- `OSReadChars`
- `OSWriteChars`
- `OSReadBytes`
- `OSWriteBytes`

## system.asynchronous\_metric\_log

Содержит исторические значения метрик из таблицы `system.asynchronous_metrics`, которые сохраняются раз в минуту. По умолчанию включена.

Столбцы:

- `event_date` ([Date](#)) — дата события.
- `event_time` ([DateTime](#)) — время события.
- `event_time_microseconds` ([DateTime64](#)) — время события в микросекундах.
- `name` ([String](#)) — название метрики.
- `value` ([Float64](#)) — значение метрики.

### Пример

```
SELECT * FROM system.asynchronous_metric_log LIMIT 10
```

event_date	event_time	name	value
2020-06-22	2020-06-22 06:57:30	jemalloc.arenas.all.pmuzzy	0
2020-06-22	2020-06-22 06:57:30	jemalloc.arenas.all.pdirty	4214
2020-06-22	2020-06-22 06:57:30	jemalloc.background_thread.run_intervals	0
2020-06-22	2020-06-22 06:57:30	jemalloc.background_thread.num_runs	0
2020-06-22	2020-06-22 06:57:30	jemalloc.retained	17657856
2020-06-22	2020-06-22 06:57:30	jemalloc.mapped	71471104
2020-06-22	2020-06-22 06:57:30	jemalloc.resident	61538304
2020-06-22	2020-06-22 06:57:30	jemalloc.metadata	6199264
2020-06-22	2020-06-22 06:57:30	jemalloc.allocated	38074336
2020-06-22	2020-06-22 06:57:30	jemalloc.epoch	2

## Смотрите также

- [system.asynchronous\\_metrics](#) — Содержит метрики, которые периодически вычисляются в фоновом режиме.
- [system.metric\\_log](#) — таблица фиксирующая историю значений метрик из `system.metrics` и `system.events`.

## system.asynchronous\_metrics

Содержит метрики, которые периодически вычисляются в фоновом режиме. Например, объём используемой оперативной памяти.

Столбцы:

- `metric` ([String](#)) — название метрики.
- `value` ([Float64](#)) — значение метрики.

## Пример

```
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

## Смотрите также

- [Мониторинг](#) — основы мониторинга в ClickHouse.
- [system.metrics](#) — таблица с мгновенно вычисляемыми метриками.
- [system.events](#) — таблица с количеством произошедших событий.
- [system.metric\\_log](#) — таблица фиксирующая историю значений метрик из `system.metrics` и `system.events`.

## system.clusters

Содержит информацию о доступных в конфигурационном файле кластерах и серверах, которые в них находятся.

Столбцы:

- `cluster` (**String**) — имя кластера.
- `shard_num` (**UInt32**) — номер шарда в кластере, начиная с 1.
- `shard_weight` (**UInt32**) — относительный вес шарда при записи данных.
- `replica_num` (**UInt32**) — номер реплики в шарде, начиная с 1.
- `host_name` (**String**) — хост, указанный в конфигурации.
- `host_address` (**String**) — ТИР-адрес хоста, полученный из DNS.
- `port` (**UInt16**) — порт для соединения с сервером.
- `is_local` (**UInt8**) — флаг, показывающий является ли хост локальным.
- `user` (**String**) — имя пользователя для соединения с сервером.
- `default_database` (**String**) — имя базы данных по умолчанию.
- `errors_count` (**UInt32**) — количество неудачных попыток хоста получить доступ к реплике.
- `slowdowns_count` (**UInt32**) — количество замен реплики из-за долгого отсутствия ответа от нее при установке соединения при хеджированных запросах.
- `estimated_recovery_time` (**UInt32**) — количество секунд до момента, когда количество ошибок будет обнулено и реплика станет доступной.

## Пример

Запрос:

```
SELECT * FROM system.clusters LIMIT 2 FORMAT Vertical;
```

Результат:

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
```

## Смотрите также

- [Table engine Distributed](#)
- [Настройка distributed\\_replica\\_error\\_cap](#)
- [Настройка distributed\\_replica\\_error\\_half\\_life](#)

## system.columns

Содержит информацию о столбцах всех таблиц.

С помощью этой таблицы можно получить информацию аналогично запросу [DESCRIBE TABLE](#), но для многих таблиц сразу.

Колонки [временных таблиц](#) содержатся в system.columns только в тех сессиях, в которых эти таблицы были созданы. Поле `database` у таких колонок пустое.

Столбцы:

- `database` ([String](#)) — имя базы данных.
- `table` ([String](#)) — имя таблицы.
- `name` ([String](#)) — имя столбца.
- `type` ([String](#)) — тип столбца.
- `position` ([UInt64](#)) — порядковый номер столбца в таблице (нумерация начинается с 1).
- `default_kind` ([String](#)) — тип выражения (`DEFAULT`, `MATERIALIZED`, `ALIAS`) для значения по умолчанию или пустая строка.
- `default_expression` ([String](#)) — выражение для значения по умолчанию или пустая строка.

- `data_compressed_bytes` (`UInt64`) — размер сжатых данных в байтах.
- `data_uncompressed_bytes` (`UInt64`) — размер распакованных данных в байтах.
- `marks_bytes` (`UInt64`) — размер засечек в байтах.
- `comment` (`String`) — комментарий к столбцу или пустая строка.
- `is_in_partition_key` (`UInt8`) — флаг, показывающий включение столбца в ключ партиционирования.
- `is_in_sorting_key` (`UInt8`) — флаг, показывающий включение столбца в ключ сортировки.
- `is_in_primary_key` (`UInt8`) — флаг, показывающий включение столбца в первичный ключ.
- `is_in_sampling_key` (`UInt8`) — флаг, показывающий включение столбца в ключ выборки.
- `compression_codec` (`String`) — имя кодека сжатия.

## Пример

```
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:
```

## system.contributors

Содержит информацию о контрибьютерах. Контрибьютеры расположены в таблице в случайном порядке. Порядок определяется заново при каждом запросе.

Столбцы:

- `name` (`String`) — Имя контрибьютера (автора коммита) из git log.

## Пример

```
SELECT * FROM system.contributors LIMIT 10
```

name
Olga Khvostikova
Max Vetrov
LiuYangkuan
svladykin
zamulla
Šimon Podlipský
BayoNet
Ilya Khomutov
Amy Krishnevsky
Loud_Scream

Чтобы найти себя в таблице, выполните запрос:

```
SELECT * FROM system.contributors WHERE name='Olga Khvostikova'
```

name
Olga Khvostikova

## system.crash\_log

Содержит информацию о трассировках стека для фатальных ошибок. Таблица не содержится в базе данных по умолчанию, а создается только при возникновении фатальных ошибок.

Колонки:

- `event_date` (`Datetime`) — Дата события.
- `event_time` (`Datetime`) — Время события.
- `timestamp_ns` (`UInt64`) — Время события с наносекундами.
- `signal` (`Int32`) — Номер сигнала, пришедшего в поток.
- `thread_id` (`UInt64`) — Идентификатор треда.
- `query_id` (`String`) — Идентификатор запроса.
- `trace` (`Array(UInt64)`) — Трассировка стека в момент ошибки. Представляет собой список физических адресов, по которым расположены вызываемые методы.
- `trace_full` (`Array(String)`) — Трассировка стека в момент ошибки. Содержит вызываемые методы.
- `version` (`String`) — Версия сервера ClickHouse.
- `revision` (`UInt32`) — Ревизия сборки сервера ClickHouse.
- `build_id` (`String`) — BuildID, сгенерированный компилятором.

## Пример

Запрос:

```
SELECT * FROM system.crash_log ORDER BY event_time DESC LIMIT 1;
```

Результат (приведён не полностью):

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:
```

## См. также

- Системная таблица [trace\\_log](#)

[Original article](#)

## system.current\_roles

Содержит активные роли текущего пользователя. `SET ROLE` изменяет содержимое этой таблицы.

Столбцы:

- `role_name` ([String](#)) — Имя роли.
- `with_admin_option` ([UInt8](#)) — Флаг, который показывает, обладает ли `current_role` роль привилегией `ADMIN OPTION`.
- `is_default` ([UInt8](#)) — Флаг, который показывает, является ли `current_role` ролью по умолчанию.

## system.data\_skipping\_indices

Содержит информацию о существующих индексах пропуска данных во всех таблицах.

Столбцы:

- `database` ([String](#)) — имя базы данных.
- `table` ([String](#)) — имя таблицы.
- `name` ([String](#)) — имя индекса.
- `type` ([String](#)) — тип индекса.
- `expr` ([String](#)) — выражение, используемое для вычисления индекса.
- `granularity` ([UInt64](#)) — количество гранул в блоке данных.

## Пример

```
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
```

Row 2:

```
database: default
table: users
name: contacts_null_idx
type: minmax
expr: assumeNotNull(contacts_null)
granularity: 1
```

## system.data\_type\_families

Содержит информацию о поддерживаемых [типах данных](#).

Столбцы:

- `name` ([String](#)) — имя типа данных.
- `case_insensitive` ([UInt8](#)) — свойство, которое показывает, зависит ли имя типа данных в запросе от регистра. Например, допустимы и `Date`, и `date`.
- `alias_to` ([String](#)) — тип данных, для которого `name` является алиасом.

### Пример

```
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

- [Синтаксис](#) — поддерживаемый SQL синтаксис.

## system.databases

Таблица содержит один столбец `name` типа `String` - имя базы данных.

Для каждой базы данных, о которой знает сервер, будет присутствовать соответствующая запись в таблице.

Эта системная таблица используется для реализации запроса `SHOW DATABASES`.

## system.detached\_parts

Содержит информацию об отсоединённых кусках таблиц семейства `MergeTree`. Столбец `reason` содержит причину, по которой кусок был отсоединен. Для кусов, отсоединённых пользователем, `reason` содержит пустую строку.

Такие куски могут быть присоединены с помощью `ALTER TABLE ATTACH PARTITION|PART`. Остальные столбцы описаны в `system.parts`.

Если имя куска некорректно, значения некоторых столбцов могут быть `NULL`. Такие куски могут быть удалены с помощью `ALTER TABLE DROP DETACHED PART`.

## system.dictionaries

Содержит информацию о [внешних словарях](#).

Столбцы:

- `database` (`String`) — Имя базы данных, в которой находится словарь, созданный с помощью DDL-запроса. Пустая строка для других словарей.
- `name` (`String`) — Имя словаря.
- `status` (`Enum8`) — Статус словаря. Возможные значения:
  - `NOT_LOADED` — Словарь не загружен, потому что не использовался.
  - `LOADED` — Словарь загружен успешно.
  - `FAILED` — Словарь не загружен в результате ошибки.
  - `LOADING` — Словарь в процессе загрузки.
  - `LOADED_AND_RELOADING` — Словарь загружен успешно, сейчас перезагружается (частые причины: запрос `SYSTEM RELOAD DICTIONARY`, таймаут, изменение настроек словаря).
  - `FAILED_AND_RELOADING` — Словарь не загружен в результате ошибки, сейчас перезагружается.
- `origin` (`String`) — Путь к конфигурационному файлу, описывающему словарь.
- `type` (`String`) — Тип размещения словаря. [Хранение словарей в памяти](#).
- `key` — **Тип ключа:** Числовой ключ (`UInt64`) или Составной ключ (`String`) — строка вида “(тип 1, тип 2, ..., тип n)”.
- `attribute.names` (`Array(String)`) — Массив [имен атрибутов](#), предоставляемых справочником.
- `attribute.types` (`Array(String)`) — Соответствующий массив [типов атрибутов](#), предоставляемых справочником.
- `bytes_allocated` (`UInt64`) — Объем оперативной памяти, используемый словарем.
- `query_count` (`UInt64`) — Количество запросов с момента загрузки словаря или с момента последней успешной перезагрузки.
- `hit_rate` (`Float64`) — Для cache-словарей — процент закэшированных значений.

- `found_rate` (`Float64`) — Процент обращений к словарю, при которых значение было найдено.
- `element_count` (`UInt64`) — Количество элементов, хранящихся в словаре.
- `load_factor` (`Float64`) — Процент заполнения словаря (для хешированного словаря — процент заполнения хэш-таблицы).
- `source` (`String`) — Текст, описывающий [источник данных](#) для словаря.
- `lifetime_min` (`UInt64`) — Минимальное [время обновления](#) словаря в памяти, по истечении которого Clickhouse попытается перезагрузить словарь (если задано `invalidate_query`, то только если он изменился). Задается в секундах.
- `lifetime_max` (`UInt64`) — Максимальное [время обновления](#) словаря в памяти, по истечении которого Clickhouse попытается перезагрузить словарь (если задано `invalidate_query`, то только если он изменился). Задается в секундах.
- `loading_start_time` (`DateTime`) — Время начала загрузки словаря.
- `loading_duration` (`Float32`) — Время, затраченное на загрузку словаря.
- `last_exception` (`String`) — Текст ошибки, возникающей при создании или перезагрузке словаря, если словарь не удалось создать.

## Пример

Настройте словарь.

```
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())
```

Убедитесь, что словарь загружен.

```
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	loading_duration	last_exception	last_successful_update_time
dictdb	dict	LOADED	dictdb.dict	Flat	UInt64	['value_default','value_expression']	['String','String']	0	1	1	0.0004887585532746823	ClickHouse: dictdb.dicttbl	0	1	2020-03-04 04:17:34	2020-03-04 04:30:34

## system.disks

Содержит информацию о дисках, заданных в [конфигурации сервера](#).

Столбцы:

- `name` (`String`) — имя диска в конфигурации сервера.
- `path` (`String`) — путь к точке монтирования в файловой системе.
- `free_space` (`UInt64`) — свободное место на диске в байтах.
- `total_space` (`UInt64`) — объём диска в байтах.
- `keep_free_space` (`UInt64`) — место, которое должно оставаться свободным на диске в байтах. Задаётся значением параметра `keep_free_space_bytes` конфигурации дисков.

## system.distributed\_ddl\_queue

Содержит информацию о [распределенных ddl запросах](#) (секция `ON CLUSTER`), которые были выполнены на кластере.

Столбцы:

- `entry` (`String`) — идентификатор запроса.
- `host_name` (`String`) — имя хоста.
- `host_address` (`String`) — IP-адрес хоста.
- `port` (`UInt16`) — порт для соединения с сервером.
- `status` (`Enum8`) — состояние запроса.
- `cluster` (`String`) — имя кластера.
- `query` (`String`) — выполненный запрос.
- `initiator` (`String`) — узел, выполнивший запрос.
- `query_start_time` (`DateTime`) — время начала запроса.
- `query_finish_time` (`DateTime`) — время окончания запроса.
- `query_duration_ms` (`UInt64`) — продолжительность выполнения запроса (в миллисекундах).
- `exception_code` (`Enum8`) — код исключения из `ZooKeeper`.

### Пример

```
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

Содержит информацию о локальных файлах, которые находятся в очереди для отправки на шарды. Эти локальные файлы содержат новые куски, которые создаются путем вставки новых данных в Distributed таблицу в асинхронном режиме.

Столбцы:

- `database` ([String](#)) — имя базы данных.
- `table` ([String](#)) — имя таблицы.
- `data_path` ([String](#)) — путь к папке с локальными файлами.
- `is_blocked` ([UInt8](#)) — флаг, указывающий на блокировку отправки локальных файлов на шарды.
- `error_count` ([UInt64](#)) — количество ошибок.
- `data_files` ([UInt64](#)) — количество локальных файлов в папке.
- `data_compressed_bytes` ([UInt64](#)) — размер сжатых данных в локальных файлах в байтах.
- `last_exception` ([String](#)) — текстовое сообщение о последней возникшей ошибке, если таковые имеются.

## Пример

```
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:
```

## Смотрите также

- [Движок таблиц Distributed](#)

## system.enabled\_roles

Содержит все активные роли на данный момент, включая текущую роль текущего пользователя и роли, назначенные для текущей роли.

Столбцы:

- `role_name` ([String](#)) — Имя роли.
- `with_admin_option` ([UInt8](#)) — Флаг, который показывает, обладает ли `enabled_role` ролью привилегией ADMIN OPTION.
- `is_current` ([UInt8](#)) — Флаг, который показывает, является ли `enabled_role` текущей ролью текущего пользователя.
- `is_default` ([UInt8](#)) — Флаг, который показывает, является ли `enabled_role` ролью по умолчанию.

## system.errors

Содержит коды ошибок с указанием количества срабатываний.

Столбцы:

- `name` ([String](#)) — название ошибки ([errorCodeToName](#)).
- `code` ([Int32](#)) — номер кода ошибки.
- `value` ([UInt64](#)) — количество ошибок.

## Пример

```
SELECT *
FROM system.errors
WHERE value > 0
ORDER BY code ASC
LIMIT 1
```

name	code	value
CANNOT_OPEN_FILE	76	1

# system.events

Содержит информацию о количестве событий, произошедших в системе. Например, в таблице можно найти, сколько запросов `SELECT` обработано с момента запуска сервера ClickHouse.

Столбцы:

- `event` (`String`) — имя события.
- `value` (`UInt64`) — количество произошедших событий.
- `description` (`String`) — описание события.

## Пример

```
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.

## Смотрите также

- [system.asynchronous\\_metrics](#) — таблица с периодически вычисляемыми метриками.
- [system.metrics](#) — таблица с мгновенно вычисляемыми метриками.
- [system.metric\\_log](#) — таблица фиксирующая историю значений метрик из `system.metrics` и `system.events`.
- [Мониторинг](#) — основы мониторинга в ClickHouse.

# system.functions

Содержит информацию об обычных и агрегатных функциях.

Столбцы:

- `name` (`String`) – Имя функции.
- `is_aggregate` (`UInt8`) – Признак, является ли функция агрегатной.

# system.grants

Привилегии пользовательских аккаунтов ClickHouse.

Столбцы:

- `user_name` (`Nullable(String)`) — Название учётной записи.
- `role_name` (`Nullable(String)`) — Роль, назначенная учетной записи пользователя.
- `access_type` (`Enum8`) — Параметры доступа для учетной записи пользователя ClickHouse.
- `database` (`Nullable(String)`) — Имя базы данных.
- `table` (`Nullable(String)`) — Имя таблицы.
- `column` (`Nullable(String)`) — Имя столбца, к которому предоставляется доступ.
- `is_partial_revoke` (`UInt8`) — Логическое значение. Показывает, были ли отменены некоторые привилегии. Возможные значения:
  - 0 — Стока описывает частичный отзыв.
  - 1 — Стока описывает грант.
- `grant_option` (`UInt8`) — Разрешение предоставлено с опцией WITH GRANT OPTION, подробнее см. [GRANT](#).

## system.graphite\_retentions

Содержит информацию о том, какие параметры `graphite_rollup` используются в таблицах с движками [\\*GraphiteMergeTree](#).

Столбцы:

- `config_name` (`String`) - Имя параметра, используемого для `graphite_rollup`.
- `regexp` (`String`) - Шаблон имени метрики.
- `function` (`String`) - Имя агрегирующей функции.
- `age` (`UInt64`) - Минимальный возраст данных в секундах.
- `precision` (`UInt64`) - Точность определения возраста данных в секундах.
- `priority` (`UInt16`) - Приоритет раздела pattern.
- `is_default` (`UInt8`) - Является ли раздел pattern дефолтным.
- `Tables.database` (`Array(String)`) - Массив имён баз данных таблиц, использующих параметр `config_name`.
- `Tables.table` (`Array(String)`) - Массив имён таблиц, использующих параметр `config_name`.

## system.licenses

Содержит информацию о лицензиях сторонних библиотек, которые находятся в директории `contrib` исходных кодов ClickHouse.

Столбцы:

- `library_name` (`String`) — Название библиотеки, к которой относится лицензия.

- `license_type` (`String`) — Тип лицензии, например, Apache, MIT.
- `license_path` (`String`) — Путь к файлу с текстом лицензии.
- `license_text` (`String`) — Текст лицензии.

## Пример

```
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

Содержит информацию о производящихся прямо сейчас слияниях и мутациях кусков для таблиц семейства MergeTree.

Столбцы:

- `database String` — Имя базы данных, в которой находится таблица.
- `table String` — Имя таблицы.
- `elapsed Float64` — Время в секундах, прошедшее от начала выполнения слияния.
- `progress Float64` — Доля выполненной работы от 0 до 1.
- `num_parts UInt64` — Количество сливаемых кусков.
- `result_part_name String` — Имя куска, который будет образован в результате слияния.
- `is_mutation UInt8` - Является ли данный процесс мутацией куска.
- `total_size_bytes_compressed UInt64` — Суммарный размер сжатых данных сливаемых кусков.
- `total_size_marks UInt64` — Суммарное количество засечек в сливаемых кусках.
- `bytes_read_uncompressed UInt64` — Количество прочитанных байт, разжатых.
- `rows_read UInt64` — Количество прочитанных строк.
- `bytes_written_uncompressed UInt64` — Количество записанных байт, несжатых.
- `rows_written UInt64` — Количество записанных строк.

# system.metric\_log

Содержит историю значений метрик из таблиц `system.metrics` и `system.events`, периодически сбрасываемую на диск.

Столбцы:

- `event_date` ([Date](#)) — дата события.
- `event_time` ([DateTime](#)) — время события.
- `event_time_microseconds` ([DateTime64](#)) — время события в микросекундах.

## Пример

```
SELECT * FROM system.metric_log LIMIT 1 FORMAT Vertical;
```

Row 1:

<code>event_date:</code>	2020-02-18
<code>event_time:</code>	2020-02-18 07:15:33
<code>milliseconds:</code>	554
<code>ProfileEvent_Query:</code>	0
<code>ProfileEvent_SelectQuery:</code>	0
<code>ProfileEvent_InsertQuery:</code>	0
<code>ProfileEvent_FileOpen:</code>	0
<code>ProfileEvent_Seek:</code>	0
<code>ProfileEvent_ReadBufferFromFileDescriptorRead:</code>	1
<code>ProfileEvent_ReadBufferFromFileDescriptorReadFailed:</code>	0
<code>ProfileEvent_ReadBufferFromFileDescriptorReadBytes:</code>	0
<code>ProfileEvent_WriteBufferFromFileDescriptorWrite:</code>	1
<code>ProfileEvent_WriteBufferFromFileDescriptorWriteFailed:</code>	0
<code>ProfileEvent_WriteBufferFromFileDescriptorWriteBytes:</code>	56
...	
<code>CurrentMetric_Query:</code>	0
<code>CurrentMetric_Merge:</code>	0
<code>CurrentMetric_PartMutation:</code>	0
<code>CurrentMetric_ReplicatedFetch:</code>	0
<code>CurrentMetric_ReplicatedSend:</code>	0
<code>CurrentMetric_ReplicatedChecks:</code>	0
...	

## Смотрите также

- [Настройка metric\\_log](#) — как включить и выключить запись истории.
- [system.asynchronous\\_metrics](#) — таблица с периодически вычисляемыми метриками.
- [system.events](#) — таблица с количеством произошедших событий.
- [system.metrics](#) — таблица с мгновенно вычисляемыми метриками.
- [Мониторинг](#) — основы мониторинга в ClickHouse.

# system.metrics

Содержит метрики, которые могут быть рассчитаны мгновенно или имеют текущее значение. Например, число одновременно обрабатываемых запросов или текущее значение задержки реплики. Эта таблица всегда актуальна.

Столбцы:

- `metric` ([String](#)) — название метрики.

- `value` ([Int64](#)) — значение метрики.
- `description` ([String](#)) — описание метрики.

Список поддерживаемых метрик смотрите в файле [src/Common/CurrentMetrics.cpp](#).

## Пример

```
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.

## Смотрите также

- [system.asynchronous\\_metrics](#) — таблица с периодически вычисляемыми метриками.
- [system.events](#) — таблица с количеством произошедших событий.
- [system.metric\\_log](#) — таблица фиксирующая историю значений метрик из `system.metrics` и `system.events`.
- [Мониторинг](#) — основы мониторинга в ClickHouse.

## system.mutations

Таблица содержит информацию о ходе выполнения [мутаций](#) таблиц семейства MergeTree. Каждой команде мутации соответствует одна строка таблицы.

Столбцы:

- `database` ([String](#)) — имя БД, к которой была применена мутация.
- `table` ([String](#)) — имя таблицы, к которой была применена мутация.
- `mutation_id` ([String](#)) — ID запроса. Для реплицированных таблиц эти ID соответствуют именам записей в директории `<table_path_in_zookeeper>/mutations/` в ZooKeeper, для нереплицированных — именам файлов в директории с данными таблицы.

- `command` ([String](#)) — команда мутации (часть запроса после `ALTER TABLE [db.]table`).
- `create_time` ([Datetime](#)) — дата и время создания мутации.
- `block_numbers.partition_id` ([Array\(String\)](#)) — Для мутаций реплицированных таблиц массив содержит номера партиций (по одной записи для каждой партиции). Для мутаций нереплицированных таблиц массив пустой.
- `block_numbers.number` ([Array\(Int64\)](#)) — Для мутаций реплицированных таблиц массив содержит по одной записи для каждой партиции, с номером блока, полученным этой мутацией. В каждой партиции будут изменены только куски, содержащие блоки с номерами меньше чем данный номер.

Для нереплицированных таблиц нумерация блоков сквозная по партициям. Поэтому массив содержит единственную запись с номером блока, полученным мутацией.

- `parts_to_do_names` ([Array\(String\)](#)) — массив с именами кусков данных, которые должны быть изменены для завершения мутации.
- `parts_to_do` ([Int64](#)) — количество кусков данных, которые должны быть изменены для завершения мутации.
- `is_done` ([UInt8](#)) — Признак, завершена ли мутация. Возможные значения:
  - 1 — мутация завершена,
  - 0 — мутация еще продолжается.

## Замечание

Даже если `parts_to_do = 0`, для реплицированной таблицы возможна ситуация, когда мутация ещё не завершена из-за долго выполняющейся операции `INSERT`, которая добавляет данные, которые нужно будет мутировать.

Если во время мутации какого-либо куска возникли проблемы, заполняются следующие столбцы:

- `latest_failed_part` ([String](#)) — имя последнего куска, мутация которого не удалась.
- `latest_fail_time` ([Datetime](#)) — дата и время последней ошибки мутации.
- `latest_fail_reason` ([String](#)) — причина последней ошибки мутации.

## См. также

- [Мутации](#)
- [Движок MergeTree](#)
- [Репликация данных](#) (семейство ReplicatedMergeTree)

## system.numbers

Таблица содержит один столбец с именем `number` типа `UInt64`, содержащим почти все натуральные числа, начиная с нуля.

Эту таблицу можно использовать для тестов, а также если вам нужно сделать перебор. Чтения из этой таблицы не распараллеливаются.

## system.numbers\_mt

То же самое, что и [system.numbers](#), но чтение распараллеливается. Числа могут возвращаться в произвольном порядке.  
Используется для тестов.

## system.one

Таблица содержит одну строку с одним столбцом `dummy` типа `UInt8`, содержащим значение 0. Эта таблица используется, если в `SELECT` запросе не указана секция `FROM`. То есть, это - аналог таблицы `DUAL`, которую можно найти в других СУБД.

## system.opentelemetry\_span\_log

Содержит информацию о `trace spans` для выполненных запросов.

Столбцы:

- `trace_id` ([UUID](#)) — идентификатор трассировки для выполненного запроса.
- `span_id` ([UInt64](#)) — идентификатор trace span.
- `parent_span_id` ([UInt64](#)) — идентификатор родительского trace span.
- `operation_name` ([String](#)) — имя операции.
- `start_time_us` ([UInt64](#)) — время начала trace span (в микросекундах).
- `finish_time_us` ([UInt64](#)) — время окончания trace span (в микросекундах).
- `finish_date` ([Date](#)) — дата окончания trace span.
- `attribute.names` ([Array\(String\)](#)) — имена атрибутов в зависимости от trace span. Заполняются согласно рекомендациям в стандарте [OpenTelemetry](#).
- `attribute.values` ([Array\(String\)](#)) — значения атрибутов в зависимости от trace span. Заполняются согласно рекомендациям в стандарте [OpenTelemetry](#).

### Пример

Запрос:

```
SELECT * FROM system.opentelemetry_span_log LIMIT 1 FORMAT Vertical;
```

Результат:

```
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: []
```

# system.part\_log

Системная таблица `system.part_log` создается только в том случае, если задана серверная настройка `part_log`.

Содержит информацию о всех событиях, произошедших с [кусками данных](#) таблиц семейства `MergeTree` (например, события добавления, удаления или слияния данных).

Столбцы:

- `query_id` ([String](#)) — идентификатор запроса `INSERT`, создавшего этот кусок.
- `event_type` ([Enum8](#)) — тип события. Столбец может содержать одно из следующих значений:
  - `NEW_PART` — вставка нового куска.
  - `MERGE_PARTS` — слияние кусков.
  - `DOWNLOAD_PART` — загрузка с реплики.
  - `REMOVE_PART` — удаление или отсоединение из таблицы с помощью [DETACH PARTITION](#).
  - `MUTATE_PART` — изменение куска.
  - `MOVE_PART` — перемещение куска между дисками.
- `event_date` ([Date](#)) — дата события.
- `event_time` ([DateTime](#)) — время события.
- `event_time_microseconds` ([DateTime64](#)) — время события с точностью до микросекунд.
- `duration_ms` ([UInt64](#)) — длительность.
- `database` ([String](#)) — имя базы данных, в которой находится кусок.
- `table` ([String](#)) — имя таблицы, в которой находится кусок.
- `part_name` ([String](#)) — имя куска.
- `partition_id` ([String](#)) — идентификатор партиции, в которую был добавлен кусок. В столбце будет значение `all`, если таблица партицируется по выражению `tuple()`.
- `path_on_disk` ([String](#)) — абсолютный путь к папке с файлами кусков данных.
- `rows` ([UInt64](#)) — число строк в куске.
- `size_in_bytes` ([UInt64](#)) — размер куска данных в байтах.
- `merged_from` ([Array\(String\)](#)) — массив имён кусков, из которых образован текущий кусок в результате слияния (также столбец заполняется в случае скачивания уже сжатого куска).
- `bytes_uncompressed` ([UInt64](#)) — количество прочитанных не сжатых байт.
- `read_rows` ([UInt64](#)) — сколько было прочитано строк при слиянии кусков.
- `read_bytes` ([UInt64](#)) — сколько было прочитано байт при слиянии кусков.
- `peak_memory_usage` ([Int64](#)) — максимальная разница между выделенной и освобождённой памятью в контексте потока.
- `error` ([UInt16](#)) — код ошибки, возникшей при текущем событии.
- `exception` ([String](#)) — текст ошибки.

Системная таблица `system.part_log` будет создана после первой вставки данных в таблицу `MergeTree`.

## Пример

```
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

Содержит информацию о кусках данных таблиц семейства `MergeTree`.

Каждая строка описывает один кусок данных.

Столбцы:

- `partition` (`String`) – имя партиции. Что такое партиция можно узнать из описания запроса `ALTER`.

Форматы:

- `YYYYMM` для автоматической схемы партиционирования по месяцам.
- `any_string` при партиционировании вручную.
- `name` (`String`) – имя куска.
- `part_type` (`String`) — формат хранения данных.

Возможные значения:

- `Wide` — каждая колонка хранится в отдельном файле.
  - `Compact` — все колонки хранятся в одном файле.
- Формат хранения данных определяется настройками `min_bytes_for_wide_part` и `min_rows_for_wide_part` таблицы `MergeTree`.
- `active` (`UInt8`) – признак активности. Если кусок активен, то он используется таблицей, в противном случае он будет удален. Неактивные куски остаются после слияний.
  - `marks` (`UInt64`) – количество засечек. Чтобы получить примерное количество строк в куске, умножьте `marks` на гранулированность индекса (обычно 8192).

- `rows` (`UInt64`) – количество строк.
- `bytes_on_disk` (`UInt64`) – общий размер всех файлов кусков данных в байтах.
- `data_compressed_bytes` (`UInt64`) – общий размер сжатой информации в куске данных. Размер всех дополнительных файлов (например, файлов с засечками) не учитывается.
- `data_uncompressed_bytes` (`UInt64`) – общий размер распакованной информации куска данных. Размер всех дополнительных файлов (например, файлов с засечками) не учитывается.
- `marks_bytes` (`UInt64`) – размер файла с засечками.
- `modification_time` (`DateTime`) – время модификации директории с куском данных. Обычно соответствует времени создания куска.
- `remove_time` (`DateTime`) – время, когда кусок стал неактивным.
- `refcount` (`UInt32`) – количество мест, в котором кусок используется. Значение больше 2 говорит о том, что кусок участвует в запросах или в слияниях.
- `min_date` (`Date`) – минимальное значение ключа даты в куске данных.
- `max_date` (`Date`) – максимальное значение ключа даты в куске данных.
- `min_time` (`DateTime`) – минимальное значение даты и времени в куске данных.
- `max_time` (`DateTime`) – максимальное значение даты и времени в куске данных.
- `partition_id` (`String`) – ID партиции.
- `min_block_number` (`UInt64`) – минимальное число кусков, из которых состоит текущий после слияния.
- `max_block_number` (`UInt64`) – максимальное число кусков, из которых состоит текущий после слияния.
- `level` (`UInt32`) – глубина дерева слияний. Если слияний не было, то `level=0`.
- `data_version` (`UInt64`) – число, которое используется для определения того, какие мутации необходимо применить к куску данных (мутации с версией большей, чем `data_version`).
- `primary_key_bytes_in_memory` (`UInt64`) – объём памяти (в байтах), занимаемой значениями первичных ключей.
- `primary_key_bytes_in_memory_allocated` (`UInt64`) – объём памяти (в байтах) выделенный для размещения первичных ключей.
- `is_frozen` (`UInt8`) – Признак, показывающий существование бэкапа партиции. 1, бэкап есть. 0, бэкапа нет. Смотрите раздел **FREEZE PARTITION**.
- `database` (`String`) – имя базы данных.
- `table` (`String`) – имя таблицы.
- `engine` (`String`) – имя движка таблицы, без параметров.
- `path` (`String`) – абсолютный путь к папке с файлами кусков данных.
- `disk` (`String`) – имя диска, на котором находится кусок данных.
- `hash_of_all_files` (`String`) – значение `sipHash128` для сжатых файлов.
- `hash_of_uncompressed_files` (`String`) – значение `sipHash128` несжатых файлов (файлы с засечками, первичным ключом и пр.)

- `uncompressed_hash_of_compressed_files` (`String`) – значение `sipHash128` данных в сжатых файлах как если бы они были разжатыми.
- `delete_ttl_info_min` (`DateTime`) — Минимальное значение ключа даты и времени для правила `TTL DELETE`.
- `delete_ttl_info_max` (`DateTime`) — Максимальное значение ключа даты и времени для правила `TTL DELETE`.
- `move_ttl_info.expression` (`Array(String)`) — Массив выражений. Каждое выражение задаёт правило `TTL MOVE`.

## Предупреждение

Массив выражений `move_ttl_info.expression` используется, в основном, для обратной совместимости. Для работы с правилами `TTL MOVE` лучше использовать поля `move_ttl_info.min` и `move_ttl_info.max`.

- `move_ttl_info.min` (`Array(DateTime)`) — Массив значений. Каждый элемент массива задаёт минимальное значение ключа даты и времени для правила `TTL MOVE`.
- `move_ttl_info.max` (`Array(DateTime)`) — Массив значений. Каждый элемент массива задаёт максимальное значение ключа даты и времени для правила `TTL MOVE`.
- `bytes` (`UInt64`) – алиас для `bytes_on_disk`.
- `marks_size` (`UInt64`) – алиас для `marks_bytes`.

## Пример

```
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
marks_bytes:       144
modification_time: 2020-06-18 13:01:49
remove_time:       0000-00-00 00:00:00
refcount:          1
min_date:          0000-00-00
max_date:          0000-00-00
min_time:          0000-00-00 00:00:00
max_time:          0000-00-00 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: 2d0657a16d9430824d35e327fcdbd87bf
hash_of_uncompressed_files: 84950cc30ba867c77a408ae21332ba29
uncompressed_hash_of_compressed_files: 1ad78f1c6843bbfb99a2c931abe7df7d
delete_ttl_info_min: 0000-00-00 00:00:00
delete_ttl_info_max: 0000-00-00 00:00:00
move_ttl_info.expression: []
move_ttl_info.min: []
move_ttl_info.max: []
```

## См. также

- [Двигок MergeTree](#)
- [TTL для столбцов и таблиц](#)

## system.parts\_columns

Содержит информацию о кусках данных и столбцах таблиц семейства [MergeTree](#).

Каждая строка описывает один кусок данных.

Столбцы:

- **partition** ([String](#)) — имя партиции. Что такое партиция вы можете узнать из описания запроса [ALTER](#).

Форматы:

- `YYYYMM` для автоматической схемы партиционирования по месяцам.
  - `any_string` при партиционировании вручную.
- **name** ([String](#)) — имя куска данных.

- `part_type` (`String`) — формат хранения данных.

Возможные значения:

- `Wide` — каждая колонка хранится в отдельном файле.
  - `Compact` — все колонки хранятся в одном файле.
- Формат хранения данных определяется настройками `min_bytes_for_wide_part` и `min_rows_for_wide_part` таблицы `MergeTree`.
- `active` (`UInt8`) — признак активности. Если кусок данных активен, то он используется таблицей, в противном случае он будет удален. Неактивные куски остаются после слияний.
  - `marks` (`UInt64`) — количество засечек. Чтобы получить примерное количество строк в куске данных, умножьте `marks` на гранулированность индекса (обычно 8192).
  - `rows` (`UInt64`) — количество строк.
  - `bytes_on_disk` (`UInt64`) — общий размер всех файлов кусков данных в байтах.
  - `data_compressed_bytes` (`UInt64`) — общий размер сжатой информации в куске данных. Размер всех дополнительных файлов (например, файлов с засечками) не учитывается.
  - `data_uncompressed_bytes` (`UInt64`) — общий размер распакованной информации в куске данных. Размер всех дополнительных файлов (например, файлов с засечками) не учитывается.
  - `marks_bytes` (`UInt64`) — размер файла с засечками.
  - `modification_time` (`DateTime`) — время модификации директории с куском данных. Обычно соответствует времени создания куска.
  - `remove_time` (`DateTime`) — время, когда кусок данных стал неактивным.
  - `refcount` (`UInt32`) — количество мест, в котором кусок данных используется. Значение больше 2 говорит о том, что кусок участвует в запросах или в слияниях.
  - `min_date` (`Date`) — минимальное значение ключа даты в куске данных.
  - `max_date` (`Date`) — максимальное значение ключа даты в куске данных.
  - `partition_id` (`String`) — ID партиции.
  - `min_block_number` (`UInt64`) — минимальное число кусков данных, из которых состоит текущий после слияния.
  - `max_block_number` (`UInt64`) — максимальное число кусков данных, из которых состоит текущий после слияния.
  - `level` (`UInt32`) — глубина дерева слияний. Если слияний не было, то `level=0`.
  - `data_version` (`UInt64`) — число, которое используется для определения того, какие мутации необходимо применить к куску данных (мутации с версией большей, чем `data_version`).
  - `primary_key_bytes_in_memory` (`UInt64`) — объём памяти в байтах, занимаемой значениями первичных ключей.
  - `primary_key_bytes_in_memory_allocated` (`UInt64`) — объём памяти в байтах, выделенный для размещения первичных ключей.
  - `database` (`String`) — имя базы данных.
  - `table` (`String`) — имя таблицы.

- `engine` (`String`) — имя движка таблицы, без параметров.
- `disk_name` (`String`) — имя диска, на котором находится кусок данных.
- `path` (`String`) — абсолютный путь к папке с файлами кусков данных.
- `column` (`String`) — имя столбца.
- `type` (`String`) — тип столбца.
- `column_position` (`UInt64`) — порядковый номер столбца (нумерация начинается с 1).
- `default_kind` (`String`) — тип выражения (`DEFAULT`, `MATERIALIZED`, `ALIAS`) для значения по умолчанию или пустая строка.
- `default_expression` (`String`) — выражение для значения по умолчанию или пустая строка.
- `column_bytes_on_disk` (`UInt64`) — общий размер столбца в байтах.
- `column_data_compressed_bytes` (`UInt64`) — общий размер сжатой информации в столбце в байтах.
- `column_data_uncompressed_bytes` (`UInt64`) — общий размер распакованной информации в столбце в байтах.
- `column_marks_bytes` (`UInt64`) — размер столбца с засечками в байтах.
- `bytes` (`UInt64`) — алиас для `bytes_on_disk`.
- `marks_size` (`UInt64`) — алиас для `marks_bytes`.

## Пример

```
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
```

## Смотрите также

- [Двигок MergeTree](#)

## system.processes

Используется для реализации запроса `SHOW PROCESSLIST`.

Столбцы:

- `user` (String) – пользователь, инициировавший запрос. При распределённом выполнении запросы отправляются на удалённые серверы от имени пользователя `default`. Поле содержит имя пользователя для конкретного запроса, а не для запроса, который инициировал этот запрос.
- `address` (String) – IP-адрес, с которого пришёл запрос. При распределённой обработке запроса аналогично. Чтобы определить откуда запрос пришел изначально, необходимо смотреть таблицу `system.processes` на сервере-источнике запроса.
- `elapsed` (Float64) – время в секундах с начала обработки запроса.
- `rows_read` (UInt64) – количество прочитанных строк. При распределённой обработке запроса на сервере-инициаторе запроса представляет собой сумму по всем удалённым серверам.
- `bytes_read` (UInt64) – количество прочитанных из таблиц байт, в несжатом виде. При распределённой обработке запроса на сервере-инициаторе запроса представляет собой сумму по всем удалённым серверам.

- `total_rows_approx` (UInt64) – приблизительная оценка общего количества строк, которые должны быть прочитаны. При распределённой обработке запроса, на сервере-инициаторе запроса, представляет собой сумму по всем удалённым серверам. Может обновляться в процессе выполнения запроса, когда становятся известны новые источники для обработки.
- `memory_usage` (UInt64) – потребление памяти запросом. Может не учитывать некоторые виды выделенной памяти. Смотрите описание настройки [max\\_memory\\_usage](#).
- `query` (String) – текст запроса. Для запросов `INSERT` не содержит вставляемые данные.
- `query_id` (String) – идентификатор запроса, если был задан.

## system.query\_log

Содержит информацию о выполняемых запросах, например, время начала обработки, продолжительность обработки, сообщения об ошибках.

### Внимание

Таблица не содержит входных данных для запросов `INSERT`.

Настойки логирования можно изменить в секции серверной конфигурации [query\\_log](#).

Можно отключить логирование настройкой `log_queries = 0`. По-возможности, не отключайте логирование, поскольку информация из таблицы важна при решении проблем.

Период сброса данных в таблицу задаётся параметром `flush_interval_milliseconds` в конфигурационной секции [query\\_log](#). Чтобы принудительно записать логи из буфера памяти в таблицу, используйте запрос [SYSTEM FLUSH LOGS](#).

ClickHouse не удаляет данные из таблица автоматически. Смотрите [Введение](#).

Таблица `system.query_log` содержит информацию о двух видах запросов:

1. Первоначальные запросы, которые были выполнены непосредственно клиентом.
2. Дочерние запросы, инициированные другими запросами (для выполнения распределенных запросов). Для дочерних запросов информация о первоначальном запросе содержится в столбцах `initial_*`.

В зависимости от статуса (столбец `type`) каждый запрос создаёт одну или две строки в таблице `query_log`:

1. Если запрос выполнен успешно, создаются два события типа `QueryStart` и `QueryFinish`.
2. Если во время обработки запроса возникла ошибка, создаются два события с типами `QueryStart` и `ExceptionWhileProcessing`.
3. Если ошибка произошла ещё до запуска запроса, создается одно событие с типом `ExceptionBeforeStart`.

Чтобы уменьшить количество запросов, регистрирующихся в таблице `query_log`, вы можете использовать настройку [log\\_queries\\_probability](#).

Столбцы:

- `type` (`Enum8`) — тип события, произошедшего при выполнении запроса. Значения:
  - `'QueryStart' = 1` — успешное начало выполнения запроса.
  - `'QueryFinish' = 2` — успешное завершение выполнения запроса.
  - `'ExceptionBeforeStart' = 3` — исключение перед началом обработки запроса.
  - `'ExceptionWhileProcessing' = 4` — исключение во время обработки запроса.
- `event_date` (`Date`) — дата начала запроса.
- `event_time` (`DateTime`) — время начала запроса.
- `event_time_microseconds` (`DateTime`) — время начала запроса с точностью до микросекунд.
- `query_start_time` (`DateTime`) — время начала обработки запроса.
- `query_start_time_microseconds` (`DateTime64`) — время начала обработки запроса с точностью до микросекунд.
- `query_duration_ms` (`UInt64`) — длительность выполнения запроса в миллисекундах.
- `read_rows` (`UInt64`) — общее количество строк, считанных из всех таблиц и табличных функций, участвующих в запросе. Включает в себя обычные подзапросы, подзапросы для `IN` и `JOIN`. Для распределенных запросов `read_rows` включает в себя общее количество строк, прочитанных на всех репликах. Каждая реплика передает собственное значение `read_rows`, а сервер-инициатор запроса суммирует все полученные и локальные значения. Объемы кэша не учитываются.
- `read_bytes` (`UInt64`) — общее количество байтов, считанных из всех таблиц и табличных функций, участвующих в запросе. Включает в себя обычные подзапросы, подзапросы для `IN` и `JOIN`. Для распределенных запросов `read_bytes` включает в себя общее количество байтов, прочитанных на всех репликах. Каждая реплика передает собственное значение `read_bytes`, а сервер-инициатор запроса суммирует все полученные и локальные значения. Объемы кэша не учитываются.
- `written_rows` (`UInt64`) — количество записанных строк для запросов `INSERT`. Для других запросов, значение столбца 0.
- `written_bytes` (`UInt64`) — объём записанных данных в байтах для запросов `INSERT`. Для других запросов, значение столбца 0.
- `result_rows` (`UInt64`) — количество строк в результате запроса `SELECT` или количество строк в запросе `INSERT`.
- `result_bytes` (`UInt64`) — объём RAM в байтах, использованный для хранения результата запроса.
- `memory_usage` (`UInt64`) — потребление RAM запросом.
- `current_database` (`String`) — имя текущей базы данных.
- `query` (`String`) — текст запроса.
- `normalized_query_hash` (`UInt64`) — идентичная хеш-сумма без значений литералов для аналогичных запросов.
- `query_kind` (`LowCardinality(String)`) — тип запроса.
- `databases` (`Array(LowCardinality(String))`) — имена баз данных, присутствующих в запросе.
- `tables` (`Array(LowCardinality(String))`) — имена таблиц, присутствующих в запросе.
- `columns` (`Array(LowCardinality(String))`) — имена столбцов, присутствующих в запросе.

- `projections` (`String`) — имена проекций, использованных при выполнении запроса.
- `exception_code` (`UInt32`) — код исключения.
- `exception` (`String`) — сообщение исключения, если запрос завершился по исключению.
- `stack_trace` (`String`) — `stack trace`. Пустая строка, если запрос успешно завершен.
- `is_initial_query` (`UInt8`) — вид запроса. Возможные значения:
  - 1 — запрос был инициирован клиентом.
  - 0 — запрос был инициирован другим запросом при выполнении распределенного запроса.
- `user` (`String`) — пользователь, запустивший текущий запрос.
- `query_id` (`String`) — ID запроса.
- `address` (`IPv6`) — IP адрес, с которого пришел запрос.
- `port` (`UInt16`) — порт, с которого клиент сделал запрос
- `initial_user` (`String`) — пользователь, запустивший первоначальный запрос (для распределенных запросов).
- `initial_query_id` (`String`) — ID родительского запроса.
- `initial_address` (`IPv6`) — IP адрес, с которого пришел родительский запрос.
- `initial_port` (`UInt16`) — порт, с которого клиент сделал родительский запрос.
- `initial_query_start_time` (`DateTime`) — время начала обработки запроса (для распределенных запросов).
- `initial_query_start_time_microseconds` (`DateTime64`) — время начала обработки запроса с точностью до микросекунд (для распределенных запросов).
- `interface` (`UInt8`) — интерфейс, с которого ушёл запрос. Возможные значения:
  - 1 — TCP.
  - 2 — HTTP.
- `os_user` (`String`) — имя пользователя операционной системы, который запустил `clickhouse-client`.
- `client_hostname` (`String`) — имя сервера, с которого присоединился `clickhouse-client` или другой TCP клиент.
- `client_name` (`String`) — `clickhouse-client` или другой TCP клиент.
- `client_revision` (`UInt32`) — ревизия `clickhouse-client` или другого TCP клиента.
- `client_version_major` (`UInt32`) — старшая версия `clickhouse-client` или другого TCP клиента.
- `client_version_minor` (`UInt32`) — младшая версия `clickhouse-client` или другого TCP клиента.
- `client_version_patch` (`UInt32`) — патч `clickhouse-client` или другого TCP клиента.
- `http_method` (`UInt8`) — HTTP метод, инициировавший запрос. Возможные значения:
  - 0 — запрос запущен с интерфейса TCP.
  - 1 — GET.
  - 2 — POST.

- `http_user_agent` (`String`) — HTTP заголовок `UserAgent`.
- `http_referer` (`String`) — HTTP заголовок `Referer` (содержит полный или частичный адрес страницы, с которой был выполнен запрос).
- `forwarded_for` (`String`) — HTTP заголовок `X-Forwarded-For`.
- `quota_key` (`String`) — ключ квоты из настроек `квот` (см. `keyed`).
- `revision` (`UInt32`) — ревизия ClickHouse.
- `ProfileEvents` (`Map(String, UInt64)`) — счетчики для изменения различных метрик. Описание метрик можно получить из таблицы `system.events` (#system\_tables-events)
- `Settings` (`Map(String, String)`) — имена настроек, которые меняются, когда клиент выполняет запрос. Чтобы разрешить логирование изменений настроек, установите параметр `log_query_settings` равным 1.
- `log_comment` (`String`) — комментарий к записи в логе. Представляет собой произвольную строку, длина которой должна быть не больше, чем `max_query_size`. Если нет комментария, то пустая строка.
- `thread_ids` (`Array(UInt64)`) — идентификаторы потоков, участвующих в обработке запросов.
- `used_aggregate_functions` (`Array(String)`) — канонические имена агрегатных функций, использованных при выполнении запроса.
- `used_aggregate_function_combinators` (`Array(String)`) — канонические имена комбинаторов агрегатных функций, использованных при выполнении запроса.
- `used_database_engines` (`Array(String)`) — канонические имена движков баз данных, использованных при выполнении запроса.
- `used_data_type_families` (`Array(String)`) — канонические имена семейств типов данных, использованных при выполнении запроса.
- `used_dictionaries` (`Array(String)`) — канонические имена источников словарей, использованных при выполнении запроса.
- `used_formats` (`Array(String)`) — канонические имена форматов, использованных при выполнении запроса.
- `used_functions` (`Array(String)`) — канонические имена функций, использованных при выполнении запроса.
- `used_storages` (`Array(String)`) — канонические имена движков таблиц, использованных при выполнении запроса.
- `used_table_functions` (`Array(String)`) — канонические имена табличных функций, использованных при выполнении запроса.

## Пример

```
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:       []
```

## Смотрите также

- [system.query\\_thread\\_log](#) — в этой таблице содержится информация о цепочке каждого выполненного запроса.

## system.query\_thread\_log

Содержит информацию о потоках, которые выполняют запросы, например, имя потока, время его запуска, продолжительность обработки запроса.

Чтобы начать логирование:

1. Настройте параметры [query\\_thread\\_log](#) в конфигурации сервера.
2. Установите значение [log\\_query\\_threads](#) равным 1.

Интервал сброса данных в таблицу задаётся параметром [flush\\_interval\\_milliseconds](#) в разделе настроек сервера [query\\_thread\\_log](#). Чтобы принудительно записать логи из буфера памяти в таблицу, используйте запрос [SYSTEM FLUSH LOGS](#).

ClickHouse не удаляет данные из таблицы автоматически. Подробности в разделе [Введение](#).

Чтобы уменьшить количество запросов, регистрирующихся в таблице [query\\_thread\\_log](#), вы можете использовать настройку [log\\_queries\\_probability](#).

Столбцы:

- `event_date` ([Date](#)) — дата завершения выполнения запроса потоком.
- `event_time` ([DateTime](#)) — дата и время завершения выполнения запроса потоком.
- `event_time_microseconds` ([DateTime](#)) — дата и время завершения выполнения запроса потоком с точностью до микросекунд.
- `query_start_time` ([DateTime](#)) — время начала обработки запроса.
- `query_start_time_microseconds` ([DateTime64](#)) — время начала обработки запроса с точностью до микросекунд.
- `query_duration_ms` ([UInt64](#)) — длительность обработки запроса в миллисекундах.
- `read_rows` ([UInt64](#)) — количество прочитанных строк.
- `read_bytes` ([UInt64](#)) — количество прочитанных байтов.
- `written_rows` ([UInt64](#)) — количество записанных строк для запросов `INSERT`. Для других запросов, значение столбца 0.
- `written_bytes` ([UInt64](#)) — объём записанных данных в байтах для запросов `INSERT`. Для других запросов, значение столбца 0.
- `memory_usage` ([Int64](#)) — разница между выделенной и освобождённой памятью в контексте потока.

- `peak_memory_usage` (`Int64`) — максимальная разница между выделенной и освобождённой памятью в контексте потока.
- `thread_name` (`String`) — имя потока.
- `thread_id` (`UInt64`) — tid (ID потока операционной системы).
- `master_thread_id` (`UInt64`) — tid (ID потока операционной системы) главного потока.
- `query` (`String`) — текст запроса.
- `is_initial_query` (`UInt8`) — вид запроса. Возможные значения:
  - 1 — запрос был инициирован клиентом.
  - 0 — запрос был инициирован другим запросом при распределенном запросе.
- `user` (`String`) — пользователь, запустивший текущий запрос.
- `query_id` (`String`) — ID запроса.
- `address` (`IPv6`) — IP адрес, с которого пришел запрос.
- `port` (`UInt16`) — порт, с которого пришел запрос.
- `initial_user` (`String`) — пользователь, запустивший первоначальный запрос (для распределенных запросов).
- `initial_query_id` (`String`) — ID родительского запроса.
- `initial_address` (`IPv6`) — IP адрес, с которого пришел родительский запрос.
- `initial_port` (`UInt16`) — порт, пришел родительский запрос.
- `interface` (`UInt8`) — интерфейс, с которого ушёл запрос. Возможные значения:
  - 1 — TCP.
  - 2 — HTTP.
- `os_user` (`String`) — имя пользователя в OS, который запустил `clickhouse-client`.
- `client_hostname` (`String`) — hostname клиентской машины, с которой присоединился `clickhouse-client` или другой TCP клиент.
- `client_name` (`String`) — `clickhouse-client` или другой TCP клиент.
- `client_revision` (`UInt32`) — ревизия `clickhouse-client` или другого TCP клиента.
- `client_version_major` (`UInt32`) — старшая версия `clickhouse-client` или другого TCP клиента.
- `client_version_minor` (`UInt32`) — младшая версия `clickhouse-client` или другого TCP клиента.
- `client_version_patch` (`UInt32`) — патч `clickhouse-client` или другого TCP клиента.
- `http_method` (`UInt8`) — HTTP метод, инициировавший запрос. Возможные значения:
  - 0 — запрос запущен с интерфейса TCP.
  - 1 — GET.
  - 2 — POST.
- `http_user_agent` (`String`) — HTTP заголовок `UserAgent`.
- `quota_key` (`String`) — «ключ квоты» из настроек `квот` (см. `keyed`).

- `revision` (`UInt32`) — ревизия ClickHouse.
- `ProfileEvents` (`Map(String, UInt64)`) — счетчики для изменения различных метрик для данного потока. Описание метрик можно получить из таблицы `system.events`.

## Пример

```
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}
```

## Смотрите также

- `system.query_log` — описание системной таблицы `query_log`, которая содержит общую информацию о выполненных запросах.

## 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':16,'SelectedRows':4,'SelectedBytes':48,'ContextLock':16,'RWLockAcquiredReadLocks':1,'RealTimeMicroseconds':698
,'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

Содержит информацию о максимумах для всех интервалов всех квот. Одной квоте могут соответствовать любое количество строк или ноль.

Столбцы:

- `quota_name` ([String](#)) — имя квоты.
- `duration` ([UInt32](#)) — длина временного интервала для расчета потребления ресурсов, в секундах.

- `is_randomized_interval` (`UInt8`) — логическое значение. Оно показывает, является ли интервал рандомизированным. Интервал всегда начинается в одно и то же время, если он не рандомизирован. Например, интервал в 1 минуту всегда начинается с целого числа минут (то есть он может начинаться в 11:20:00, но никогда не начинается в 11:20:01), интервал в один день всегда начинается в полночь UTC. Если интервал рандомизирован, то самый первый интервал начинается в произвольное время, а последующие интервалы начинаются один за другим. Значения:
  - 0 — интервал рандомизирован.
  - 1 — интервал не рандомизирован.
- `max_queries` (`Nullable(UInt64)`) — максимальное число запросов.
- `max_query_selects` (`Nullable(UInt64)`) — максимальное число запросов `SELECT`.
- `max_query_inserts` (`Nullable(UInt64)`) — максимальное число запросов `INSERT`.
- `max_errors` (`Nullable(UInt64)`) — максимальное количество ошибок.
- `max_result_rows` (`Nullable(UInt64)`) — максимальное количество строк результата.
- `max_result_bytes` (`Nullable(UInt64)`) — максимальный объем оперативной памяти в байтах, используемый для хранения результата запроса.
- `max_read_rows` (`Nullable(UInt64)`) — максимальное количество строк, считываемых из всех таблиц и табличных функций, участвующих в запросе.
- `max_read_bytes` (`Nullable(UInt64)`) — максимальное количество байтов, считываемых из всех таблиц и табличных функций, участвующих в запросе.
- `max_execution_time` (`Nullable(Float64)`) — максимальное время выполнения запроса, в секундах.

## system.quota\_usage

Использование квоты текущим пользователем: сколько используется и сколько осталось.

Столбцы:

- `quota_name` (`String`) — имя квоты.
- `quota_key` (`String`) — значение ключа. Например, если `keys = ip_address`, `quota_key` может иметь значение '192.168.1.1'.
- `start_time` (`Nullable(DateTime)`) — время начала расчета потребления ресурсов.
- `end_time` (`Nullable(DateTime)`) — время окончания расчета потребления ресурсов.
- `duration` (`Nullable(UInt64)`) — длина временного интервала для расчета потребления ресурсов, в секундах.
- `queries` (`Nullable(UInt64)`) — общее количество запросов на этом интервале.
- `query_selects` (`Nullable(UInt64)`) — общее количество запросов `SELECT` на этом интервале.
- `query_inserts` (`Nullable(UInt64)`) — общее количество запросов `INSERT` на этом интервале.
- `max_queries` (`Nullable(UInt64)`) — максимальное количество запросов.
- `errors` (`Nullable(UInt64)`) — число запросов, вызвавших ошибки.
- `max_errors` (`Nullable(UInt64)`) — максимальное число ошибок.

- `result_rows` (`Nullable(UInt64)`) — общее количество строк результата.
- `max_result_rows` (`Nullable(UInt64)`) — максимальное количество строк результата.
- `result_bytes` (`Nullable(UInt64)`) — объем оперативной памяти в байтах, используемый для хранения результата запроса.
- `max_result_bytes` (`Nullable(UInt64)`) — максимальный объем оперативной памяти, используемый для хранения результата запроса, в байтах.
- `read_rows` (`Nullable(UInt64)`) — общее число исходных строк, считываемых из таблиц для выполнения запроса на всех удаленных серверах.
- `max_read_rows` (`Nullable(UInt64)`) — максимальное количество строк, считываемых из всех таблиц и табличных функций, участвующих в запросах.
- `read_bytes` (`Nullable(UInt64)`) — общее количество байт, считанных из всех таблиц и табличных функций, участвующих в запросах.
- `max_read_bytes` (`Nullable(UInt64)`) — максимальное количество байт, считываемых из всех таблиц и табличных функций.
- `execution_time` (`Nullable(Float64)`) — общее время выполнения запроса, в секундах.
- `max_execution_time` (`Nullable(Float64)`) — максимальное время выполнения запроса.

## Смотрите также

- [SHOW QUOTA](#)

---

## system.quotas

Содержит информацию о [квотах](#).

Столбцы:

- `name` (`String`) — Имя квоты.
- `id` (`UUID`) — ID квоты.
- `storage(String)` — Хранилище квот. Возможные значения: "users.xml", если квота задана в файле users.xml, "disk" — если квота задана в SQL-запросе.

- `keys` (`Array(Enum8)`) — Ключ определяет совместное использование квоты. Если два соединения используют одну и ту же квоту, они совместно используют один и тот же объем ресурсов.
- Значения:
- `[]` — Все пользователи используют одну и ту же квоту.
  - `['user_name']` — Соединения с одинаковым именем пользователя используют одну и ту же квоту.
  - `['ip_address']` — Соединения с одинаковым IP-адресом используют одну и ту же квоту.
  - `['client_key']` — Соединения с одинаковым ключом используют одну и ту же квоту. Ключ может быть явно задан клиентом. При использовании `clickhouse-client`, передайте ключевое значение в параметре `--quota_key`, или используйте параметр `quota_key` файле настроек клиента. В случае использования HTTP интерфейса, используйте заголовок `X-ClickHouse-Quota`.
  - `['user_name', 'client_key']` — Соединения с одинаковым ключом используют одну и ту же квоту. Если ключ не предоставлен клиентом, то квота отслеживается для `user_name`.
  - `['client_key', 'ip_address']` — Соединения с одинаковым ключом используют одну и ту же квоту. Если ключ не предоставлен клиентом, то квота отслеживается для `ip_address`.
- `durations` (`Array(UInt64)`) — Длины временных интервалов для расчета потребления ресурсов, в секундах.
  - `apply_to_all` (`UInt8`) — Логическое значение. Он показывает, к каким пользователям применяется квота. Значения:
    - `0` — Квота применяется к пользователям, перечисленным в списке `apply_to_list`.
    - `1` — Квота применяется к пользователям, за исключением тех, что перечислены в списке `apply_to_except`.
  - `apply_to_list` (`Array(String)`) — Список имен пользователей/ролей к которым применяется квота.
  - `apply_to_except` (`Array(String)`) — Список имен пользователей/ролей к которым квота применяться не должна.

## Смотрите также

- [SHOW QUOTAS](#)

## system.quotas\_usage

Использование квот всеми пользователями.

Столбцы:

- `quota_name` (`String`) — имя квоты.
- `quota_key` (`String`) — ключ квоты.
- `is_current` (`UInt8`) — квота используется для текущего пользователя.
- `start_time` (`Nullable(DateTime)`) — время начала расчета потребления ресурсов.
- `end_time` (`Nullable(DateTime)`) — время окончания расчета потребления ресурсов.
- `duration` (`Nullable(UInt32)`) — длина временного интервала для расчета потребления ресурсов, в секундах.
- `queries` (`Nullable(UInt64)`) — общее количество запросов на этом интервале.

- `max_queries` (`Nullable(UInt64)`) — максимальное число запросов.
- `query_selects` (`Nullable(UInt64)`) — общее количество запросов `SELECT` на этом интервале.
- `max_query_selects` (`Nullable(UInt64)`) — максимальное количество запросов `SELECT` на этом интервале.
- `query_inserts` (`Nullable(UInt64)`) — общее количество запросов `INSERT` на этом интервале.
- `max_query_inserts` (`Nullable(UInt64)`) — максимальное количество запросов `INSERT` на этом интервале.
- `errors` (`Nullable(UInt64)`) — число запросов, вызвавших ошибки.
- `max_errors` (`Nullable(UInt64)`) — максимальное число ошибок.
- `result_rows` (`Nullable(UInt64)`) — общее количество строк, приведенных в результате.
- `max_result_rows` (`Nullable(UInt64)`) — максимальное количество исходных строк, считываемых из таблиц.
- `result_bytes` (`Nullable(UInt64)`) — объем оперативной памяти в байтах, используемый для хранения результата запроса.
- `max_result_bytes` (`Nullable(UInt64)`) — максимальный объем оперативной памяти, используемый для хранения результата запроса, в байтах.
- `read_rows` (`Nullable(UInt64)`) — общее число исходных строк, считываемых из таблиц для выполнения запроса на всех удаленных серверах.
- `max_read_rows` (`Nullable(UInt64)`) — максимальное количество строк, считываемых из всех таблиц и табличных функций, участвующих в запросах.
- `read_bytes` (`Nullable(UInt64)`) — общее количество байт, считанных из всех таблиц и табличных функций, участвующих в запросах.
- `max_read_bytes` (`Nullable(UInt64)`) — максимальное количество байт, считываемых из всех таблиц и табличных функций.
- `execution_time` (`Nullable(Float64)`) — общее время выполнения запроса, в секундах.
- `max_execution_time` (`Nullable(Float64)`) — максимальное время выполнения запроса.

## Смотрите также

- [SHOW QUOTA](#)

## system.replicas

Содержит информацию и статус для реплицируемых таблиц, расположенных на локальном сервере.

Эту таблицу можно использовать для мониторинга. Таблица содержит по строчке для каждой Replicated\*-таблицы.

Пример:

```
SELECT *
FROM system.replicas
WHERE table = 'test_table'
FORMAT Vertical
```

Query id: dc6dcdbc-dc28-4df9-ae27-4354f5b3b13e

Row 1:

```
database:          db
table:            test_table
engine:           ReplicatedMergeTree
is_leader:        1
canBecomeLeader:  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}
```

Столбцы:

- **database** (String) - имя БД.
- **table** (String) - имя таблицы.
- **engine** (String) - имя движка таблицы.
- **is\_leader** (UInt8) - является ли реплика лидером.  
Несколько реплик могут быть лидерами одновременно. Реплике можно запретить быть лидером с помощью `merge_tree` настройки `replicated_can_become_leader`. Лидеры назначают фоновые слияния, которые следует произвести.  
Замечу, что запись можно осуществлять на любую реплику (доступную и имеющую сессию в ZK), независимо от лидерства.
- **can\_become\_leader** (UInt8) - может ли реплика быть лидером.
- **is\_READONLY** (UInt8) - находится ли реплика в режиме «только для чтения»  
Этот режим включается, если в конфиге нет секции с ZK; если при переинициализации сессии в ZK произошла неизвестная ошибка; во время переинициализации сессии с ZK.
- **is\_SESSION\_EXPIRED** (UInt8) - истекла ли сессия с ZK. В основном, то же самое, что и `is_READONLY`.
- **future\_parts** (UInt32) - количество кусков с данными, которые появятся в результате INSERT-ов или слияний, которых ещё предстоит сделать
- **parts\_to\_check** (UInt32) - количество кусков с данными в очереди на проверку. Кусок помещается в очередь на проверку, если есть подозрение, что он может быть битым.

- `zookeeper_path` (`String`) - путь к данным таблицы в ZK.
- `replica_name` (`String`) - имя реплики в ZK; разные реплики одной таблицы имеют разное имя.
- `replica_path` (`String`) - путь к данным реплики в ZK. То же самое, что конкатенация `zookeeper_path/replicas/replica_path`.
- `columns_version` (`Int32`) - номер версии структуры таблицы. Обозначает, сколько раз был сделан ALTER. Если на репликах разные версии, значит некоторые реплики сделали ещё не все ALTER-ы.
- `queue_size` (`UInt32`) - размер очереди действий, которые предстоит сделать. К действиям относятся вставки блоков данных, слияния, и некоторые другие действия. Как правило, совпадает с `future_parts`.
- `inserts_in_queue` (`UInt32`) - количество вставок блоков данных, которые предстоит сделать. Обычно вставки должны быстро реплицироваться. Если величина большая - значит что-то не так.
- `merges_in_queue` (`UInt32`) - количество слияний, которые предстоит сделать. Бывают длинные слияния - то есть, это значение может быть больше нуля продолжительное время.
- `part_mutations_in_queue` (`UInt32`) - количество мутаций, которые предстоит сделать.
- `queue_oldest_time` (`DateTime`) - если `queue_size` больше 0, показывает, когда была добавлена в очередь самая старая операция.
- `inserts_oldest_time` (`DateTime`) - см. `queue_oldest_time`.
- `merges_oldest_time` (`DateTime`) - см. `queue_oldest_time`.
- `part_mutations_oldest_time` (`DateTime`) - см. `queue_oldest_time`.

Следующие 4 столбца имеют ненулевое значение только если активна сессия с ZK.

- `log_max_index` (`UInt64`) - максимальный номер записи в общем логе действий.
- `log_pointer` (`UInt64`) - максимальный номер записи из общего лога действий, которую реплика скопировала в свою очередь для выполнения, плюс единица. Если `log_pointer` сильно меньше `log_max_index`, значит что-то не так.
- `last_queue_update` (`DateTime`) - время последнего обновления запроса.
- `absolute_delay` (`UInt64`) - задержка (в секундах) для текущей реплики.
- `total_replicas` (`UInt8`) - общее число известных реплик этой таблицы.
- `active_replicas` (`UInt8`) - число реплик этой таблицы, имеющих сессию в ZK; то есть, число работающих реплик.
- `last_queue_update_exception` (`String`) - если в очереди есть битые записи. Особенно важно, когда в ClickHouse нарушается обратная совместимость между версиями, а записи журнала, сделанные более новыми версиями, не могут быть проанализированы старыми версиями.
- `zookeeper_exception` (`String`) - последнее сообщение об исключении. Появляется, если ошибка произошла при получении информации из ZooKeeper.
- `replica_is_active` (`Map(String, UInt8)`) — соответствие между именем реплики и признаком активности реплики.

Если запрашивать все столбцы, то таблица может работать слегка медленно, так как на каждую строчку делается несколько чтений из ZK.

Если не запрашивать последние 4 столбца (log\_max\_index, log\_pointer, total\_replicas, active\_replicas), то таблица работает быстро.

Например, так можно проверить, что всё хорошо:

```
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
```

Если этот запрос ничего не возвращает - значит всё хорошо.

## system.replicated\_fetches

Содержит информацию о выполняемых в данный момент фоновых операциях скачивания кусков данных с других реплик.

Столбцы:

- `database` (`String`) — имя базы данных.
- `table` (`String`) — имя таблицы.
- `elapsed` (`Float64`) — время, прошедшее от момента начала скачивания куска, в секундах.
- `progress` (`Float64`) — доля выполненной работы от 0 до 1.
- `result_part_name` (`String`) — имя скачиваемого куска.
- `result_part_path` (`String`) — абсолютный путь к скачиваемому куску.
- `partition_id` (`String`) — идентификатор партиции.
- `total_size_bytes_compressed` (`UInt64`) — общий размер сжатой информации в скачиваемом куске в байтах.
- `bytes_read_compressed` (`UInt64`) — размер сжатой информации, считанной из скачиваемого куска, в байтах.

- `source_replica_path` (`String`) — абсолютный путь к исходной реплике.
- `source_replica_hostname` (`String`) — имя хоста исходной реплики.
- `source_replica_port` (`UInt16`) — номер порта исходной реплики.
- `interserver_scheme` (`String`) — имя межсерверной схемы.
- `URI` (`String`) — универсальный идентификатор ресурса.
- `to_detached` (`UInt8`) — флаг, указывающий на использование выражения `TO DETACHED` в текущих фоновых операциях.
- `thread_id` (`UInt64`) — идентификатор потока.

## Пример

```
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
```

## Смотрите также

- [Управление таблицами ReplicatedMergeTree](#)

# system.replication\_queue

Содержит информацию о задачах из очередей репликации, хранящихся в ZooKeeper, для таблиц семейства `ReplicatedMergeTree`.

Столбцы:

- `database` (`String`) — имя базы данных.
- `table` (`String`) — имя таблицы.
- `replica_name` (`String`) — имя реплики в ZooKeeper. Разные реплики одной и той же таблицы имеют различные имена.
- `position` (`UInt32`) — позиция задачи в очереди.
- `node_name` (`String`) — имя узла в ZooKeeper.

- `type` (`String`) — тип задачи в очереди:
  - `GET_PART` — скачать кусок с другой реплики.
  - `ATTACH_PART` — присоединить кусок. Задача может быть выполнена и с куском из нашей собственной реплики (если он находится в папке `detached`). Эта задача практически идентична задаче `GET_PART`, лишь немного оптимизирована.
  - `MERGE_PARTS` — выполнить слияние кусков.
  - `DROP_RANGE` — удалить куски в партициях из указанного диапазона.
  - `CLEAR_COLUMN` — удалить указанный столбец из указанной партиции. Примечание: не используется с 20.4.
  - `CLEAR_INDEX` — удалить указанный индекс из указанной партиции. Примечание: не используется с 20.4.
  - `REPLACE_RANGE` — удалить указанный диапазон кусков и заменить их на новые.
  - `MUTATE_PART` — применить одну или несколько мутаций к куску.
  - `ALTER_METADATA` — применить изменения структуры таблицы в результате запросов с выражением `ALTER`.
- `create_time` (`Datetime`) — дата и время отправки задачи на выполнение.
- `required_quorum` (`UInt32`) — количество реплик, ожидающих завершения задачи, с подтверждением о завершении. Этот столбец актуален только для задачи `GET_PARTS`.
- `source_replica` (`String`) — имя исходной реплики.
- `new_part_name` (`String`) — имя нового куска.
- `parts_to_merge` (`Array (String)`) — имена кусков, которые требуется смергить или обновить.
- `is_detach` (`UInt8`) — флаг, указывающий на присутствие в очереди задачи `DETACH_PARTS`.
- `is_currently_executing` (`UInt8`) — флаг, указывающий на выполнение конкретной задачи на данный момент.
- `num_tries` (`UInt32`) — количество неудачных попыток выполнить задачу.
- `last_exception` (`String`) — текст сообщения о последней возникшей ошибке, если таковые имеются.
- `last_attempt_time` (`Datetime`) — дата и время последней попытки выполнить задачу.
- `num_postponed` (`UInt32`) — количество отложенных задач.
- `postpone_reason` (`String`) — причина, по которой была отложена задача.
- `last_postpone_time` (`Datetime`) — дата и время, когда была отложена задача в последний раз.
- `merge_type` (`String`) — тип текущего слияния. Пусто, если это мутация.

## Пример

```
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' doesn't 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
```

## Смотрите также

- [Управление таблицами ReplicatedMergeTree](#)

## system.role\_grants

Содержит **гранты** ролей для пользователей и ролей. Чтобы добавить записи в эту таблицу, используйте команду GRANT role TO user.

Столбцы:

- `user_name` (`Nullable(String)`) — Имя пользователя.
- `role_name` (`Nullable(String)`) — Имя роли.
- `granted_role_name` (`String`) — Имя роли, назначенной для роли `role_name`. Чтобы назначить одну роль другой используйте GRANT role1 TO role2.
- `granted_role_is_default` (`UInt8`) — Флаг, который показывает, является ли `granted_role` ролью по умолчанию. Возможные значения:
  - 1 — `granted_role` является ролью по умолчанию.
  - 0 — `granted_role` не является ролью по умолчанию.
- `with_admin_option` (`UInt8`) — Флаг, который показывает, обладает ли `granted_role` роль привилегией ADMIN OPTION. Возможные значения:
  - 1 — Роль обладает привилегией ADMIN OPTION.
  - 0 — Роль не обладает привилегией ADMIN OPTION.

## system.roles

Содержит сведения о **ролях**.

Столбцы:

- `name` (`String`) — Имя роли.

- `id` (**UUID**) — ID роли.
- `storage` (**String**) — Путь к хранилищу ролей. Настраивается в параметре `access_control_path`.

## Смотрите также

- [SHOW ROLES](#)

## system.row\_policies

Содержит фильтры безопасности уровня строк (политики строк) для каждой таблицы, а также список ролей и/или пользователей, к которым применяются эти политики.

Столбцы:

- `name` (**String**) — Имя политики строк.
- `short_name` (**String**) — Короткое имя политики строк. Имена политик строк являются составными, например: `myfilter ON mydb.mytable`. Здесь `myfilter ON mydb.mytable` — это имя политики строк, `myfilter` — короткое имя.
- `database` (**String**) — Имя базы данных.
- `table` (**String**) — Имя таблицы.
- `id` (**UUID**) — ID политики строк.
- `storage` (**String**) — Имя каталога, в котором хранится политика строк.
- `select_filter` (**Nullable(String)**) — Условие, которое используется для фильтрации строк.
- `is_restrictive` (**UInt8**) — Показывает, ограничивает ли политика строк доступ к строкам, подробнее см. [CREATE ROW POLICY](#). Значения:
  - 0 — Политика строк определяется с помощью условия 'AS PERMISSIVE'.
  - 1 — Политика строк определяется с помощью условия 'AS RESTRICTIVE'.
- `apply_to_all` (**UInt8**) — Показывает, что политики строк заданы для всех ролей и/или пользователей.
- `apply_to_list` (**Array(String)**) — Список ролей и/или пользователей, к которым применяется политика строк.
- `apply_to_except` (**Array(String)**) — Политики строк применяются ко всем ролям и/или пользователям, за исключением перечисленных.

## Смотрите также

- [SHOW POLICIES](#)

## system.settings

Содержит информацию о сессионных настройках для текущего пользователя.

Столбцы:

- `name` (**String**) — имя настройки.
- `value` (**String**) — значение настройки.

- `changed` (`UInt8`) — показывает, изменена ли настройка по отношению к значению по умолчанию.
- `description` (`String`) — краткое описание настройки.
- `min` (`Nullable(String)`) — минимальное значение настройки, если задано ограничение. Если нет, то поле содержит `NULL`.
- `max` (`Nullable(String)`) — максимальное значение настройки, если задано ограничение. Если нет, то поле содержит `NULL`.
- `readonly` (`UInt8`) — Показывает, может ли пользователь изменять настройку:
  - `0` — Текущий пользователь может изменять настройку.
  - `1` — Текущий пользователь не может изменять настройку.

## Пример

Пример показывает как получить информацию о настройках, имена которых содержат `min_i`.

```
SELECT *
FROM system.settings
WHERE name LIKE '%min_i%'
```

name	value	changed	description
	min	max	readonly
min_insert_block_size_rows	1048576	0   Squash blocks passed to INSERT query to specified size in rows, if blocks are not big enough.	NULL   NULL   0
min_insert_block_size_bytes	268435456	0   Squash blocks passed to INSERT query to specified size in bytes, if blocks are not big enough.	NULL   NULL   0
read_backoff_min_interval_between_events_ms	1000	0   Settings to reduce the number of threads in case of slow reads. Do not pay attention to the event, if the previous one has passed less than a certain amount of time.	NULL   NULL   0

Использование `WHERE changed` может быть полезно, например, если необходимо проверить:

- Что настройки корректно загрузились из конфигурационного файла и используются.
- Настройки, изменённые в текущей сессии.

```
SELECT * FROM system.settings WHERE changed AND name='load_balancing'
```

## См. также

- [Настройки](#)
- [Разрешения для запросов](#)
- [Ограничения для значений настроек](#)
- Выражение `SHOW SETTINGS`

## system.settings\_profile\_elements

Описывает содержимое профиля настроек:

- Ограничения.

- Роли и пользователи, к которым применяется настройка.
- Родительские профили настроек.

Столбцы:

- `profile_name` (`Nullable(String)`) — Имя профиля настроек.
- `user_name` (`Nullable(String)`) — Имя пользователя.
- `role_name` (`Nullable(String)`) — Имя роли.
- `index` (`UInt64`) — Порядковый номер элемента профиля настроек.
- `setting_name` (`Nullable(String)`) — Имя настройки.
- `value` (`Nullable(String)`) — Значение настройки.
- `min` (`Nullable(String)`) — Минимальное значение настройки. `NONE` если не задано.
- `max` (`Nullable(String)`) — Максимальное значение настройки. `NONE` если не задано.
- `readonly` (`Nullable(UInt8)`) — Профиль разрешает только запросы на чтение.
- `inherit_profile` (`Nullable(String)`) — Родительский профиль для данного профиля настроек. `NONE` если не задано. Профиль настроек может наследовать все значения и ограничения настроек (`min`, `max`, `readonly`) от своего родительского профиля.

## system.settings\_profiles

Содержит свойства сконфигурированных профилей настроек.

Столбцы:

- `name` (`String`) — Имя профиля настроек.
- `id` (`UUID`) — ID профиля настроек.
- `storage` (`String`) — Путь к хранилищу профилей настроек. Настраивается в параметре `access_control_path`.
- `num_elements` (`UInt64`) — Число элементов для этого профиля в таблице `system.settings_profile_elements`.
- `apply_to_all` (`UInt8`) — Признак, который показывает, что параметры профиля заданы для всех ролей и/или пользователей.
- `apply_to_list` (`Array(String)`) — Список ролей и/или пользователей, к которым применяется профиль настроек.
- `apply_to_except` (`Array(String)`) — Профиль настроек применяется ко всем ролям и/или пользователям, за исключением перечисленных.

## Смотрите также

- [SHOW PROFILES](#)

## system.stack\_trace

Содержит трассировки стека всех серверных потоков. Позволяет разработчикам анализировать состояние сервера.

Для анализа логов используйте [функции интроспекции](#): `addressToLine`, `addressToSymbol` и `demangle`.

Столбцы:

- `thread_name` ([String](#)) — имя потока.
- `thread_id` ([UInt64](#)) — идентификатор потока.
- `query_id` ([String](#)) — идентификатор запроса. Может быть использован для получения подробной информации о выполненном запросе из системной таблицы [query\\_log](#).
- `trace` ([Array\(UInt64\)](#)) — [трассировка стека](#). Представляет собой список физических адресов, по которым расположены вызываемые методы.

## Пример

Включение функций интроспекции:

```
SET allow_introspection_functions = 1;
```

Получение символов из объектных файлов ClickHouse:

```
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::executeImpl(DB::QueryPipeline&, std::__1::shared_ptr<DB::IBlockInputStream> const&, std::__1::optional<DB::Pipe>)
DB::InterpreterSelectQuery::execute()
DB::InterpreterSelectWithUnionQuery::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
```

Получение имен файлов и номеров строк в исходном коде ClickHouse:

```
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
```

## Смотрите также

- [Функции интроспекции](#) — описание функций интроспекции и примеры использования.
- [system.trace\\_log](#) — системная таблица, содержащая трассировки стека, собранные профилировщиком выборочных запросов.
- [arrayMap](#) — описание и пример использования функции arrayMap.
- [arrayFilter](#) — описание и пример использования функции arrayFilter.

## system.storage\_policies

Содержит информацию о политиках хранения и томах, заданных в [конфигурации сервера](#).

Столбцы:

- `policy_name` ([String](#)) — имя политики хранения.
- `volume_name` ([String](#)) — имя тома, который содержится в политике хранения.
- `volume_priority` ([UInt64](#)) — порядковый номер тома согласно конфигурации.
- `disks` ([Array\(String\)](#)) — имена дисков, содержащихся в политике хранения.
- `max_data_part_size` ([UInt64](#)) — максимальный размер куска данных, который может храниться на дисках тома (0 — без ограничений).
- `move_factor` — доля доступного свободного места на томе, если места становится меньше, то данные начнут перемещение на следующий том, если он есть (по умолчанию 0.1).
- `prefer_not_to_merge` ([UInt8](#)) — Значение настройки `prefer_not_to_merge`. Если данная настройка включена, то слияние данных, хранящихся на данном томе, не допускается. Это позволяет контролировать работу ClickHouse с медленными дисками.

Если политика хранения содержит несколько томов, то каждому тому соответствует отдельная запись в таблице.

## system.table\_engines

Содержит информацию про движки таблиц, поддерживаемые сервером, а также об их возможностях.

Эта таблица содержит следующие столбцы (тип столбца показан в скобках):

- `name` (String) — имя движка.
- `supports_settings` (UInt8) — флаг, показывающий поддержку секции SETTINGS.
- `supports_skipping_indices` (UInt8) — флаг, показывающий поддержку индексов пропуска данных.
- `supports_ttl` (UInt8) — флаг, показывающий поддержку TTL.
- `supports_sort_order` (UInt8) — флаг, показывающий поддержку секций PARTITION\_BY, PRIMARY\_KEY, ORDER\_BY и SAMPLE\_BY.
- `supports_replication` (UInt8) — флаг, показывающий поддержку репликации.
- `supports_deduplication` (UInt8) — флаг, показывающий наличие в движке дедупликации данных.

Пример:

```
SELECT *
FROM system.table_engines
WHERE name in ('Kafka', 'MergeTree', 'ReplicatedCollapsingMergeTree')
```

name	supports_settings	supports_skipping_indices	supports_sort_order	supports_ttl	supports_replication	supports_deduplication
Kafka	1	0	0	0	1	0
MergeTree	1	1	1	1	0	0
ReplicatedCollapsingMergeTree	1	1	1	1	1	1

### Смотрите также

- [Секции движка](#) семейства MergeTree
- [Настройки Kafka](#)
- [Настройки Join](#)

## system.tables

Содержит метаданные каждой таблицы, о которой знает сервер.

Отсоединённые таблицы ([DETACH](#)) не отображаются в system.tables.

Информация о [временных таблицах](#) содержится в system.tables только в тех сессиях, в которых эти таблицы были созданы. Поле `database` у таких таблиц пустое, а флаг `is_temporary` включен.

Столбцы:

- `database` (`String`) — имя базы данных, в которой находится таблица.
- `name` (`String`) — имя таблицы.
- `engine` (`String`) — движок таблицы (без параметров).
- `is_temporary` (`UInt8`) — флаг, указывающий на то, временная это таблица или нет.
- `data_path` (`String`) — путь к данным таблицы в файловой системе.
- `metadata_path` (`String`) — путь к табличным метаданным в файловой системе.
- `metadata_modification_time` (`DateTime`) — время последней модификации табличных метаданных.
- `dependencies_database` (`Array(String)`) — зависимости базы данных.
- `dependencies_table` (`Array(String)`) — табличные зависимости (таблицы `MaterializedView`, созданные на базе текущей таблицы).
- `create_table_query` (`String`) — запрос, при помощи которого создавалась таблица.
- `engine_full` (`String`) — параметры табличного движка.
- `partition_key` (`String`) — ключ partitionирования таблицы.
- `sorting_key` (`String`) — ключ сортировки таблицы.
- `primary_key` (`String`) - первичный ключ таблицы.
- `sampling_key` (`String`) — ключ сэмплирования таблицы.
- `storage_policy` (`String`) - политика хранения данных:
  - `MergeTree`
  - `Distributed`
- `total_rows` (`Nullable(UInt64)`) - общее количество строк, если есть возможность быстро определить точное количество строк в таблице, в противном случае `NULL` (включая базовую таблицу `Buffer`).
- `total_bytes` (`Nullable(UInt64)`) - общее количество байт, если можно быстро определить точное количество байт для таблицы на накопителе, в противном случае `NULL` (не включает в себя никакого базового хранилища).
  - Если таблица хранит данные на диске, возвращает используемое пространство на диске (т. е. сжатое).
  - Если таблица хранит данные в памяти, возвращает приблизительное количество используемых байт в памяти.
- `lifetime_rows` (`Nullable(UInt64)`) - общее количество строк, добавленных оператором `INSERT` с момента запуска сервера (только для таблиц `Buffer`).
- `lifetime_bytes` (`Nullable(UInt64)`) - общее количество байт, добавленных оператором `INSERT` с момента запуска сервера (только для таблиц `Buffer`).
- `comment` (`String`) — комментарий к таблице.

Таблица `system.tables` используется при выполнении запроса `SHOW TABLES`.

## Пример

```
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
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

Содержит записи логов. Уровень логирования для таблицы может быть ограничен параметром сервера text\_log.level.

Столбцы:

- `event_date` (Date) — дата создания записи.
- `event_time` (DateTime) — время создания записи.
- `event_time_microseconds` (DateTime) — время создания записи с точностью до микросекунд.
- `microseconds` (UInt32) — время создания записи в микросекундах.

- `thread_name` (String) — название потока, из которого была сделана запись.
- `thread_id` (UInt64) — идентификатор потока ОС.
- `level` (Enum8) — уровень логирования записи. Возможные значения:
  - 1 или 'Fatal'.
  - 2 или 'Critical'.
  - 3 или 'Error'.
  - 4 или 'Warning'.
  - 5 или 'Notice'.
  - 6 или 'Information'.
  - 7 или 'Debug'.
  - 8 или 'Trace'.
- `query_id` (String) — идентификатор запроса.
- `logger_name` (LowCardinality(String)) — название логгера (DDLWorker).
- `message` (String) — само тело записи.
- `revision` (UInt32) — ревизия ClickHouse.
- `source_file` (LowCardinality(String)) — исходный файл, из которого была сделана запись.
- `source_line` (UInt64) — исходная строка, из которой была сделана запись.

## Пример

```
SELECT * FROM system.text_log LIMIT 1 \G
```

Row 1:

```
event_date:      2020-09-10
event_time:     2020-09-10 11:23:07
event_time_microseconds: 2020-09-10 11:23:07.871397
microseconds:    871397
thread_name:    clickhouse-serv
thread_id:      564917
level:          Information
query_id:
logger_name:    DNSCacheUpdater
message:        Update period 15 seconds
revision:       54440
source_file:    /ClickHouse/src/Interpreters/DNSCacheUpdater.cpp; void DB::DNSCacheUpdater::start()
source_line:    45
```

## system.time\_zones

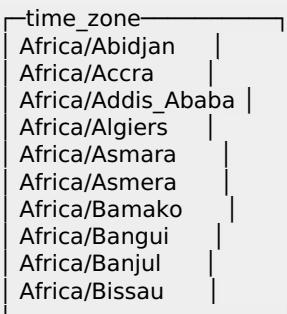
Contains a list of time zones that are supported by the ClickHouse server. This list of timezones might vary depending on the version of ClickHouse.

Columns:

- `time_zone` (String) — List of supported time zones.

## Example

```
SELECT * FROM system.time_zones LIMIT 10
```



## system.trace\_log

Содержит экземпляры трассировок стека адресов вызова, собранные с помощью семпленирующего профайлера запросов.

ClickHouse создает эту таблицу когда установлена настройка `trace_log` в конфигурационном файле сервера. А также настройки `query_profiler_real_time_period_ns` и `query_profiler_cpu_time_period_ns`.

Для анализа stack traces, используйте функции интроспекции `addressToLine`, `addressToSymbol` и `demangle`.

Столбцы:

- `event_date(Date)` — дата в момент снятия экземпляра стэка адресов вызова.
- `event_time(DateTime)` — дата и время в момент снятия экземпляра стэка адресов вызова.
- `event_time_microseconds (DateTime64)` — дата и время в момент снятия экземпляра стэка адресов вызова с точностью до микросекунд.
- `revision(UInt32)` — ревизия сборки сервера ClickHouse.

Во время соединения с сервером через `clickhouse-client`, вы видите строку похожую на `Connected to ClickHouse server version 19.18.1 revision 54429.`. Это поле содержит номер после `revision`, но не содержит строку после `version`.

- `trace_type(Enum8)` — тип трассировки:
  - `Real` — сбор трассировок стека адресов вызова по времени wall-clock.
  - `CPU` — сбор трассировок стека адресов вызова по времени CPU.
  - `Memory` — сбор выделенной памяти, когда ее размер превышает относительный инкремент.
  - `MemorySample` — сбор случайно выделенной памяти.
- `thread_number(UInt32)` — идентификатор треда.
- `query_id(String)` — идентификатор запроса который может быть использован для получения деталей о запросе из таблицы `query_log` system table.
- `trace(Array(UInt64))` — трассировка стека адресов вызова в момент семплирования. Каждый элемент массива — это адрес виртуальной памяти внутри процесса сервера ClickHouse.

## Пример

```
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

Содержит список [аккаунтов пользователей](#), настроенных на сервере.

Столбцы:

- `name` ([String](#)) — Имя пользователя.
- `id` ([UUID](#)) — ID пользователя.
- `storage` ([String](#)) — Путь к хранилищу пользователей. Настраивается в параметре `access_control_path`.
- `auth_type` ([Enum8](#)('no\_password' = 0,'plaintext\_password' = 1, 'sha256\_password' = 2, 'double\_sha1\_password' = 3)) — Показывает тип аутентификации. Существует несколько способов идентификации пользователя: без пароля, с помощью обычного текстового пароля, с помощью шифрования [SHA256] (<https://ru.wikipedia.org/wiki/SHA-2>) или с помощью шифрования [double SHA-1] (<https://ru.wikipedia.org/wiki/SHA-1>).
- `auth_params` ([String](#)) — Параметры аутентификации в формате JSON, зависят от `auth_type`.
- `host_ip` ([Array\(String\)](#)) — IP-адреса хостов, которым разрешено подключаться к серверу ClickHouse.
- `host_names` ([Array\(String\)](#)) — Имена хостов, которым разрешено подключаться к серверу ClickHouse.
- `host_names_regexp` ([Array\(String\)](#)) — Регулярное выражение для имен хостов, которым разрешено подключаться к серверу ClickHouse.
- `host_names_like` ([Array\(String\)](#)) — Имена хостов, которым разрешено подключаться к серверу ClickHouse, заданные с помощью предиката LIKE.
- `default_roles_all` ([UInt8](#)) — Показывает, что все предоставленные роли установлены для пользователя по умолчанию.
- `default_roles_list` ([Array\(String\)](#)) — Список предоставленных ролей по умолчанию.
- `default_roles_except` ([Array\(String\)](#)) — Все предоставленные роли задаются по умолчанию, за исключением перечисленных.

## Смотрите также

- [SHOW USERS](#)

# system.zookeeper

Таблицы не существует, если ZooKeeper не сконфигурирован. Позволяет читать данные из ZooKeeper кластера, описанного в конфигурации.

В запросе обязательно в секции WHERE должно присутствовать условие на равенство path - путь в ZooKeeper, для детей которого вы хотите получить данные.

Запрос `SELECT * FROM system.zookeeper WHERE path = '/clickhouse'` выведет данные по всем детям узла `/clickhouse`.

Чтобы вывести данные по всем узлам в корне, напишите `path = '/'`.

Если узла, указанного в path не существует, то будет брошено исключение.

Столбцы:

- `name String` — Имя узла.
- `path String` — Путь к узлу.
- `value String` — Значение узла.
- `dataLength Int32` — Размер значения.
- `numChildren Int32` — Количество детей.
- `czxid Int64` — Идентификатор транзакции, в которой узел был создан.
- `mzxid Int64` — Идентификатор транзакции, в которой узел был последний раз изменён.
- `pzxid Int64` — Идентификатор транзакции, последний раз удаливший или добавивший детей.
- `ctime DateTime` — Время создания узла.
- `mtime DateTime` — Время последней модификации узла.
- `version Int32` — Версия узла - количество раз, когда узел был изменён.
- `cversion Int32` — Количество добавлений или удалений детей.
- `aversion Int32` — Количество изменений ACL.
- `ephemeralOwner Int64` — Для эфемерных узлов - идентификатор сессии, которая владеет этим узлом.

Пример:

```
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

Эта таблица содержит информацию о параметрах запроса к серверу ZooKeeper и ответа от него.

Для запросов заполняются только столбцы с параметрами запроса, а остальные столбцы заполняются значениями по умолчанию (0 или NULL). Когда поступает ответ, данные добавляются в столбцы с параметрами ответа на запрос.

Столбцы с параметрами запроса:

- **type** (**Enum**) — тип события в клиенте ZooKeeper. Может иметь одно из следующих значений:
  - **Request** — запрос отправлен.
  - **Response** — ответ получен.
  - **Finalize** — соединение разорвано, ответ не получен.
- **event\_date** (**Date**) — дата, когда произошло событие.
- **event\_time** (**DateTime64**) — дата и время, когда произошло событие.
- **address** (**IPv6**) — IP адрес сервера ZooKeeper, с которого был сделан запрос.
- **port** (**UInt16**) — порт сервера ZooKeeper, с которого был сделан запрос.
- **session\_id** (**Int64**) — идентификатор сессии, который сервер ZooKeeper создает для каждого соединения.
- **xid** (**Int32**) — идентификатор запроса внутри сессии. Обычно это последовательный номер запроса, одинаковый у строки запроса и у парной строки **response/finalize**.
- **has\_watch** (**UInt8**) — установлен ли запрос **watch**.

- `op_num` (**Enum**) — тип запроса или ответа на запрос.
- `path` (**String**) — путь к узлу ZooKeeper, указанный в запросе. Пустая строка, если запрос не требует указания пути.
- `data` (**String**) — данные, записанные на узле ZooKeeper (для запросов `SET` и `CREATE` — что запрос хотел записать, для ответа на запрос `GET` — что было прочитано), или пустая строка.
- `is_ephemeral` (**UInt8**) — создается ли узел ZooKeeper как **ephemeral**.
- `is_sequential` (**UInt8**) — создается ли узел ZooKeeper как **sequential**.
- `version` (**Nullable(Int32)**) — версия узла ZooKeeper, которую запрос ожидает увидеть при выполнении. Поддерживается для запросов `CHECK`, `SET`, `REMOVE` (-1 — запрос не проверяет версию, `NULL` — для других запросов, которые не поддерживают проверку версии).
- `requests_size` (**UInt32**) — количество запросов, включенных в мультизапрос (это специальный запрос, который состоит из нескольких последовательных обычных запросов, выполняющихся атомарно). Все запросы, включенные в мультизапрос, имеют одинаковый `xid`.
- `request_idx` (**UInt32**) — номер запроса, включенного в мультизапрос (0 — для мультизапроса, далее по порядку с 1).

Столбцы с параметрами ответа на запрос:

- `xid` (**Int64**) — идентификатор транзакции в ZooKeeper. Последовательный номер, выданный сервером ZooKeeper в ответе на успешно выполненный запрос (0 — запрос не был выполнен, возвращена ошибка или клиент ZooKeeper не знает, был ли выполнен запрос).
- `error` (**Nullable(Enum)**) — код ошибки. Может иметь много значений, здесь приведены только некоторые из них:
  - `ZOK` — запрос успешно выполнен.
  - `ZCONNECTIONLOSS` — соединение разорвано.
  - `ZOPERATIONTIMEOUT` — истекло время ожидания выполнения запроса.
  - `ZSESSIONEXPIRED` — истекло время сессии.
  - `NULL` — выполнен запрос.
- `watch_type` (**Nullable(Enum)**) — тип события `watch` (для ответов на запрос при `op_num = Watch`), для остальных ответов: `NULL`.
- `watch_state` (**Nullable(Enum)**) — статус события `watch` (для ответов на запрос при `op_num = Watch`), для остальных ответов: `NULL`.
- `path_created` (**String**) — путь к созданному узлу ZooKeeper (для ответов на запрос `CREATE`). Может отличаться от `path`, если узел создается как `sequential`.
- `stat_czid` (**Int64**) — идентификатор транзакции, в результате которой был создан узел ZooKeeper.
- `stat_mxid` (**Int64**) — идентификатор транзакции, которая последней модифицировала узел ZooKeeper.
- `stat_pzid` (**Int64**) — идентификатор транзакции, которая последней модифицировала дочерние узлы ZooKeeper.
- `stat_version` (**Int32**) — количество изменений в данных узла ZooKeeper.
- `stat_cversion` (**Int32**) — количество изменений в дочерних узлах ZooKeeper.

- `stat_dataLength` (`Int32`) — длина поля данных узла ZooKeeper.
- `stat_numChildren` (`Int32`) — количество дочерних узлов ZooKeeper.
- `children` (`Array(String)`) — список дочерних узлов ZooKeeper (для ответов на запрос `LIST`).

## Пример

Запрос:

```
SELECT * FROM system.zookeeper_log WHERE (session_id = '106662742089334927') AND (xid = '10858') FORMAT  
Vertical;
```

Результат:

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']
```

## См. также

- [ZooKeeper](#)
- [Руководство по ZooKeeper](#)

# 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"  
1000000000
```

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/>

## Конфигурационные параметры сервера

Раздел содержит описания настроек сервера, которые не могут изменяться на уровне сессии или запроса.

Рассмотренные настройки хранятся в файле `config.xml` сервера ClickHouse.

Прочие настройки описаны в разделе «[Настройки](#)».

Перед изучением настроек ознакомьтесь с разделом [Конфигурационные файлы](#), обратите внимание на использование подстановок (атрибуты `incl` и `optional`).

## Конфигурационные параметры сервера `builtin_dictionaries_reload_interval`

Интервал (в секундах) перезагрузки встроенных словарей.

ClickHouse перезагружает встроенные словари с заданным интервалом. Это позволяет править словари «на лету» без перезапуска сервера.

Значение по умолчанию - 3600.

### Пример

```
<builtin_dictionaries_reload_interval>3600</builtin_dictionaries_reload_interval>
```

## compression

Настройки компрессии данных.

### Внимание

Лучше не использовать, если вы только начали работать с ClickHouse.

Общий вид конфигурации:

```
<compression>
  <case>
    <min_part_size>...</min_part_size>
    <min_part_size_ratio>...</min_part_size_ratio>
    <method>...</method>
    <level>...</level>
  </case>
  ...
</compression>
```

Поля блока `<case>`:

- `min_part_size` - Минимальный размер части таблицы.
- `min_part_size_ratio` - Отношение размера минимальной части таблицы к полному размеру таблицы.
- `method` - Метод сжатия. Возможные значения: `lz4`, `lz4hc`, `zstd`.
- `level` – Уровень сжатия. См. [Кодеки](#).

Можно сконфигурировать несколько разделов `<case>`.

ClickHouse проверяет условия для `min_part_size` и `min_part_size_ratio` и выполнит те блоки `case`, для которых условия совпали.

- Если кусок данных совпадает с условиями, ClickHouse использует указанные метод сжатия.
- Если кусок данных совпадает с несколькими блоками `case`, ClickHouse использует первый совпавший блок условий.

Если ни один `<case>` не подходит, то ClickHouse применит алгоритм сжатия `lz4`.

## Пример

```
<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>
```

## encryption

Настраивает команду для получения ключа, используемого [кодеками шифрования](#). Ключ (или несколько ключей) должен быть записан в переменные окружения или установлен в конфигурационном файле.

Ключи могут быть представлены в шестнадцатеричной или строковой форме. Их длина должна быть равна 16 байтам.

## Пример

Загрузка из файла конфигурации:

```
<encryption_codecs>
  <aes_128_gcm_siv>
    <key>12345567812345678</key>
  </aes_128_gcm_siv>
</encryption_codecs>
```

## Примечание

Хранение ключей в конфигурационном файле не рекомендовано. Это не безопасно. Вы можете переместить ключи в отдельный файл на секретном диске и сделать `symlink` к этому конфигурационному файлу в папке `config.d/`.

Загрузка из файла конфигурации, когда ключ представлен в шестнадцатеричной форме:

```
<encryption_codecs>
  <aes_128_gcm_siv>
    <key_hex>00112233445566778899aabbccddeeff</key_hex>
  </aes_128_gcm_siv>
</encryption_codecs>
```

Загрузка ключа из переменной окружения:

```
<encryption_codecs>
  <aes_128_gcm_siv>
    <key_hex from_env="KEY"></key_hex>
  </aes_128_gcm_siv>
</encryption_codecs>
```

Параметр `current_key_id` устанавливает текущий ключ для шифрования, и все указанные ключи можно использовать для расшифровки.

Все эти методы могут быть применены для нескольких ключей:

```
<encryption_codecs>
  <aes_128_gcm_siv>
    <key_hex id="0">00112233445566778899aabbccddeeff</key_hex>
    <key_hex id="1" from_env="."></key_hex>
    <current_key_id>1</current_key_id>
  </aes_128_gcm_siv>
</encryption_codecs>
```

Параметр `current_key_id` указывает текущий ключ для шифрования.

Также пользователь может добавить одноразовое случайное число длинной 12 байт (по умолчанию шифрование и дешифровка будут использовать одноразовое число длинной 12 байт, заполненное нулями):

```
<encryption_codecs>
  <aes_128_gcm_siv>
    <nonce>0123456789101</nonce>
  </aes_128_gcm_siv>
</encryption_codecs>
```

Одноразовое число также может быть представлено в шестнадцатеричной форме:

```
<encryption_codecs>
  <aes_128_gcm_siv>
    <nonce_hex>abcdefabcdef</nonce_hex>
  </aes_128_gcm_siv>
</encryption_codecs>
```

Всё вышеперечисленное также применимо для алгоритма `aes_256_gcm_siv` (но ключ должен быть длиной 32 байта).

## custom\_settings\_prefixes

Список префиксов для [пользовательских настроек](#). Префиксы должны перечисляться через запятую.

### Пример

```
<custom_settings_prefixes>custom_</custom_settings_prefixes>
```

## **См. также**

- [Пользовательские настройки](#)

## **core\_dump**

Задает мягкое ограничение для размера файла дампа памяти.

Возможные значения:

- положительное целое число.

Значение по умолчанию: 1073741824 (1 ГБ).

### **Примечание**

Жесткое ограничение настраивается с помощью системных инструментов.

## **Пример**

```
<core_dump>
  <size_limit>1073741824</size_limit>
</core_dump>
```

## **database\_atomic\_delay\_before\_drop\_table\_sec**

Устанавливает задержку перед удалением табличных данных, в секундах. Если запрос имеет идентификатор SYNC, эта настройка игнорируется.

Значение по умолчанию: 480 (8 минут).

## **default\_database**

База данных по умолчанию.

Перечень баз данных можно получить запросом [SHOW DATABASES](#).

## **Пример**

```
<default_database>default</default_database>
```

## **default\_profile**

Профиль настроек по умолчанию.

Профили настроек находятся в файле, указанном в параметре `user_config`.

## **Пример**

```
<default_profile>default</default_profile>
```

## **default\_replica\_path**

Путь к таблице в ZooKeeper.

## **Пример**

```
<default_replica_path>/clickhouse/tables/{uuid}/{shard}</default_replica_path>
```

## default\_replica\_name

Имя реплики в ZooKeeper.

### Пример

```
<default_replica_name>{replica}</default_replica_name>
```

## dictionaries\_config

Путь к конфигурации внешних словарей.

Путь:

- Указывается абсолютным или относительно конфигурационного файла сервера.
- Может содержать wildcard-ы \* и ?.

Смотрите также «[Внешние словари](#)».

### Пример

```
<dictionaries_config>*_dictionary.xml</dictionaries_config>
```

## dictionaries\_lazy\_load

Отложенная загрузка словарей.

Если `true`, то каждый словарь создаётся при первом использовании. Если словарь не удалось создать, то вызов функции, использующей словарь, сгенерирует исключение.

Если `false`, то все словари создаются при старте сервера, если словарь или словари создаются слишком долго или создаются с ошибкой, то сервер загружается без этих словарей и продолжает попытки создать эти словари.

По умолчанию - `true`.

### Пример

```
<dictionaries_lazy_load>true</dictionaries_lazy_load>
```

## format\_schema\_path

Путь к каталогу со схемами для входных данных. Например со схемами для формата [CapnProto](#).

### Пример

```
<!-- Directory containing schema files for various input formats. -->
<format_schema_path>format_schemas/</format_schema_path>
```

## graphite

Отправка данных в [Graphite](#).

## Настройки:

- host – Сервер Graphite.
- port – Порт сервера Graphite.
- interval – Период отправки в секундах.
- timeout – Таймаут отправки данных в секундах.
- root\_path – Префикс для ключей.
- metrics – Отправка данных из таблицы **system.metrics**.
- events – Отправка дельты данных, накопленной за промежуток времени из таблицы **system.events**.
- events\_cumulative – Отправка суммарных данных из таблицы **system.events**.
- asynchronous\_metrics – Отправка данных из таблицы **system.asynchronous\_metrics**.

Можно определить несколько секций `<graphite>`, например, для передачи различных данных с различной частотой.

## Пример

```
<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

Настройка прореживания данных для Graphite.

Подробнее читайте в разделе [GraphiteMergeTree](#).

## Пример

```
<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

Порт для обращений к серверу по протоколу HTTP(s).

Если указан `https_port`, то требуется конфигурирование [openSSL](#).

Если указан `http_port`, то настройка openSSL игнорируется, даже если она задана.

### Пример

```
<https_port>9999</https_port>
```

## http\_server\_default\_response

Страница, показываемая по умолчанию, при обращении к HTTP(s) серверу ClickHouse.  
Значение по умолчанию «Ok.» (с переводом строки на конце).

### Пример

Показывает <https://tabix.io/> при обращении к `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>
```

## include\_from

Путь к файлу с подстановками.

Подробности смотрите в разделе «[Конфигурационные файлы](#)».

### Пример

```
<include_from>/etc/metrica.xml</include_from>
```

## interserver\_http\_port

Порт для обмена между серверами ClickHouse.

### Пример

```
<interserver_http_port>9009</interserver_http_port>
```

## interserver\_http\_host

Имя хоста, которое могут использовать другие серверы для обращения к этому хосту.

Если не указано, то определяется аналогично команде `hostname -f`.

Удобно использовать, чтобы отвязаться от конкретного сетевого интерфейса.

### Пример

```
<interserver_http_host>example.yandex.ru</interserver_http_host>
```

## interserver\_https\_port

Порт для обмена данными между репликами ClickHouse по протоколу HTTPS.

## Пример

```
<interserver_https_port>9010</interserver_https_port>
```

## interserver\_https\_host

Имя хоста, которое могут использовать другие реплики для обращения к нему по протоколу HTTPS.

## Пример

```
<interserver_https_host>example.yandex.ru</interserver_https_host>
```

## interserver\_http\_credentials

Имя пользователя и пароль, использующиеся для аутентификации при [репликации движками Replicated\\*](#). Это имя пользователя и пароль используются только для взаимодействия между репликами кластера и никак не связаны с аутентификацией клиентов ClickHouse. Сервер проверяет совпадение имени и пароля для соединяющихся с ним реплик, а также использует это же имя и пароль для соединения с другими репликами. Соответственно, эти имя и пароль должны быть прописаны одинаковыми для всех реплик кластера.

По умолчанию аутентификация не используется.

## Примечание

Эти учетные данные являются общими для обмена данными по протоколам HTTP и HTTPS.

Раздел содержит следующие параметры:

- `user` — имя пользователя.
- `password` — пароль.

## Пример конфигурации

```
<interserver_http_credentials>
  <user>admin</user>
  <password>222</password>
</interserver_http_credentials>
```

## keep\_alive\_timeout

Время в секундах, в течение которого ClickHouse ожидает входящих запросов прежде чем закрыть соединение. Значение по умолчанию: 10 секунд.

## Пример

```
<keep_alive_timeout>10</keep_alive_timeout>
```

## listen\_host

Ограничение по хостам, с которых может прийти запрос. Если необходимо, чтобы сервер отвечал всем, то надо указать `::`.

Примеры:

```
<listen_host>::1</listen_host>
<listen_host>127.0.0.1</listen_host>
```

## logger

Настройки логирования.

Ключи:

- `level` - Уровень логирования. Допустимые значения: `trace`, `debug`, `information`, `warning`, `error`.
- `log` - Файл лога. Содержит все записи согласно `level`.
- `errorlog` - Файл лога ошибок.
- `size` - Размер файла. Действует для `log` и `errorlog`. Как только файл достиг размера `size`, ClickHouse архивирует и переименовывает его, а на его месте создает новый файл лога.
- `count` - Количество заархивированных файлов логов, которые сохраняет ClickHouse.

### Пример

```
<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>
```

Также, существует поддержка записи в `syslog`. Пример настроек:

```
<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>
```

Ключи для `syslog`:

- `use_syslog` - обязательная настройка, если требуется запись в `syslog`
- `address` - хост[:порт] демона `syslogd`. Если не указан, используется локальный
- `hostname` - дополнительно, имя хоста, с которого отсылаются логи
- `facility` - **категория `syslog`**, записанная в верхнем регистре, с префиксом «`LOG_`»: (`LOG_USER`, `LOG_DAEMON`, `LOG_LOCAL3` и прочие).  
Значения по умолчанию: при указанном `address` - `LOG_USER`, иначе - `LOG_DAEMON`
- `format` - формат сообщений. Возможные значения - `bsd` и `syslog`

## send\_crash\_reports

Настройки для отправки сообщений о сбоях в команду разработчиков ядра ClickHouse через [Sentry](#). Включение этих настроек, особенно в pre-production среде, может дать очень ценную информацию и поможет развитию ClickHouse.

Сервер на котором включены данные настройки должен иметь доступ в Интернет по протоколу IPv4 (на момент написания документации IPv6 не поддерживается публичным облаком Sentry) для правильной работы данной функциональности.

Ключи:

- `enabled` – Булевый флаг чтобы включить функциональность, по умолчанию `false`. Установите `true` чтобы разрешить отправку отчетов о сбоях.
- `endpoint` – Вы можете переопределить URL на который будут отсылааться отчеты об ошибках и использовать собственную инсталляцию Sentry. Используйте URL синтаксис [Sentry DSN](#).
- `anonymize` - Запретить отсылку имени хоста сервера в отчете о сбое.
- `http_proxy` - Настройка HTTP proxy для отсылки отчетов о сбоях.
- `debug` - Настроить клиентскую библиотеку Sentry в debug режим.
- `tmp_path` - Путь в файловой системе для временного хранения состояния отчетов о сбоях перед отправкой на сервер Sentry.

## Рекомендованные настройки

```
<send_crash_reports>
  <enabled>true</enabled>
</send_crash_reports>
```

## macros

Подстановки параметров реплицируемых таблиц.

Можно не указывать, если реплицируемые таблицы не используются.

Подробнее смотрите в разделе [Создание реплицируемых таблиц](#).

### Пример

```
<macros incl="macros" optional="true" />
```

## mark\_cache\_size

Приблизительный размер (в байтах) кэша засечек, используемых движками таблиц семейства [MergeTree](#).

Кэш общий для сервера, память выделяется по мере необходимости.

### Пример

```
<mark_cache_size>5368709120</mark_cache_size>
```

## max\_server\_memory\_usage

Ограничивает объём оперативной памяти, используемой сервером ClickHouse. Настройка может быть задана только для профиля `default`.

Возможные значения:

- Положительное целое число.
- 0 — автоматически.

Значение по умолчанию: 0.

## Дополнительная информация

Значение по умолчанию для `max_server_memory_usage` рассчитывается как `memory_amount * max_server_memory_usage_to_ram_ratio`.

### См. также

- [max\\_memory\\_usage](#)
- [max\\_server\\_memory\\_usage\\_to\\_ram\\_ratio](#)

## max\_server\_memory\_usage\_to\_ram\_ratio

Определяет долю оперативной памяти, доступную для использования сервером Clickhouse. Если сервер попытается использовать больше, предоставляемый ему объём памяти будет ограничен до расчётного значения.

Возможные значения:

- Положительное число с плавающей запятой.
- 0 — сервер Clickhouse может использовать всю оперативную память.

Значение по умолчанию: 0.9.

### Использование

На серверах с небольшим объёмом оперативной памяти и файла подкачки может потребоваться установить настройку `max_server_memory_usage_to_ram_ratio` в значение, большее 1.

### Пример

```
<max_server_memory_usage_to_ram_ratio>0.9</max_server_memory_usage_to_ram_ratio>
```

### См. также

- [max\\_server\\_memory\\_usage](#)

## max\_concurrent\_queries

Определяет максимальное количество одновременно обрабатываемых запросов, связанных с таблицей семейства MergeTree. Запросы также могут быть ограничены настройками:  
[max\\_concurrent\\_queries\\_for\\_user](#), [max\\_concurrent\\_queries\\_for\\_all\\_users](#),  
[min\\_marks\\_to\\_honor\\_max\\_concurrent\\_queries](#).

### Примечание

Параметры этих настроек могут быть изменены во время выполнения запросов и вступят в силу немедленно. Запросы, которые уже запущены, выполняются без изменений.

Возможные значения:

- Положительное целое число.
- 0 — выключена.

### Пример

```
<max_concurrent_queries>100</max_concurrent_queries>
```

## max.concurrent\_queries\_for\_user

Определяет максимальное количество одновременно обрабатываемых запросов, связанных с таблицей семейства MergeTree, для пользователя.

Возможные значения:

- Положительное целое число.
- 0 — выключена.

### Пример

```
<max_concurrent_queries_for_user>5</max_concurrent_queries_for_user>
```

## max.concurrent\_queries\_for\_all\_users

Если значение этой настройки меньше или равно текущему количеству одновременно обрабатываемых запросов, то будет сгенерировано исключение.

Пример: max.concurrent\_queries\_for\_all\_users установлен на 99 для всех пользователей. Чтобы выполнять запросы даже когда сервер перегружен, администратор баз данных устанавливает для себя значение настройки на 100.

Изменение настройки для одного запроса или пользователя не влияет на другие запросы.

Значение по умолчанию: 0 — отсутствие ограничений.

### Пример

```
<max_concurrent_queries_for_all_users>99</max_concurrent_queries_for_all_users>
```

### Смотрите также

- [max.concurrent\\_queries](#)

## min\_marks\_to\_honor\_max\_concurrent\_queries

Определяет минимальное количество засечек, считываемых запросом для применения настройки [max.concurrent\\_queries](#).

Возможные значения:

- Положительное целое число.
- 0 — выключена.

### Пример

```
<min_marks_to_honor_max_concurrent_queries>10</min_marks_to_honor_max_concurrent_queries>
```

## max\_connections

Максимальное количество входящих соединений.

### Пример

```
<max_connections>4096</max_connections>
```

## max\_open\_files

Максимальное количество открытых файлов.

По умолчанию - `maximum`.

Рекомендуется использовать в Mac OS X, поскольку функция `getrlimit()` возвращает некорректное значение.

### Пример

```
<max_open_files>262144</max_open_files>
```

## max\_table\_size\_to\_drop

Ограничение на удаление таблиц.

Если размер таблицы семейства **MergeTree** превышает `max_table_size_to_drop` (в байтах), то ее нельзя удалить запросом `DROP`.

Если таблицу все же необходимо удалить, не перезапуская при этом сервер ClickHouse, то необходимо создать файл `<clickhouse-path>/flags/force_drop_table` и выполнить запрос `DROP`.

Значение по умолчанию - `50GB`.

Значение `0` означает, что можно удалять все таблицы без ограничений.

### Пример

```
<max_table_size_to_drop>0</max_table_size_to_drop>
```

## max\_thread\_pool\_size

Максимальное количество потоков в глобальном пуле потоков.

Значение по умолчанию: `10000`.

### Пример

```
<max_thread_pool_size>12000</max_thread_pool_size>
```

## merge\_tree

Тонкая настройка таблиц семейства **MergeTree**.

Подробнее смотрите в заголовочном файле `MergeTreeSettings.h`.

## Пример

```
<merge_tree>
  <max_suspicious_broken_parts>5</max_suspicious_broken_parts>
</merge_tree>
```

## metric\_log

Эта настройка включена по умолчанию. Если это не так, вы можете включить ее сами.

### Включение

Чтобы вручную включить сбор истории метрик в таблице `system.metric_log`, создайте `/etc/clickhouse-server/config.d/metric_log.xml` следующего содержания:

```
<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>
```

### Выключение

Чтобы отключить настройку `metric_log`, создайте файл `/etc/clickhouse-server/config.d/disable_metric_log.xml` следующего содержания:

```
<clickhouse>
  <metric_log remove="1" />
</clickhouse>
```

## replicated\_merge\_tree

Тонкая настройка таблиц в [ReplicatedMergeTree](#).

Эта настройка имеет более высокий приоритет.

Подробнее смотрите в заголовочном файле `MergeTreeSettings.h`.

## Пример

```
<replicated_merge_tree>
  <max_suspicious_broken_parts>5</max_suspicious_broken_parts>
</replicated_merge_tree>
```

## openSSL

Настройки клиента/сервера SSL.

Поддержку SSL обеспечивает библиотека `libpoco`. Описание интерфейса находится в файле [SSLManager.h](#)

Ключи настроек сервера/клиента:

- `privateKeyFile` - Путь к файлу с секретным ключом сертификата в формате PEM. Файл может содержать ключ и сертификат одновременно.

- certificateFile - Путь к файлу сертификата клиента/сервера в формате PEM. Можно не указывать, если `privateKeyFile` содержит сертификат.
- caConfig - Путь к файлу или каталогу, которые содержат доверенные корневые сертификаты.
- verificationMode - Способ проверки сертификатов узла. Подробности находятся в описании класса `Context`. Допустимые значения: `none`, `relaxed`, `strict`, `once`.
- verificationDepth - Максимальная длина верификационной цепи. Верификация завершится ошибкой, если длина цепи сертификатов превысит установленное значение.
- loadDefaultCAFile - Признак того, что будут использоваться встроенные CA-сертификаты для OpenSSL. Допустимые значения: `true`, `false`. |
- cipherList - Поддерживаемые OpenSSL-шифры. Например, `ALL:!ADH:!LOW:!EXP:!MD5:@STRENGTH`.
- cacheSessions - Включение/выключение кеширования сессии. Использовать обязательно вместе с `sessionIdContext`. Допустимые значения: `true`, `false`.
- sessionIdContext - Уникальный набор произвольных символов, которые сервер добавляет к каждому сгенерированному идентификатору. Длина строки не должна превышать `SSL_MAX_SSL_SESSION_ID_LENGTH`. Рекомендуется к использованию всегда, поскольку позволяет избежать проблем как в случае, если сервер кеширует сессию, так и если клиент затребовал кеширование. По умолчанию  `${application.name}`.
- sessionCacheSize - Максимальное количество сессий, которые кэширует сервер. По умолчанию - `1024*20`. 0 - неограниченное количество сессий.
- sessionTimeout - Время кеширования сессии на сервере.
- extendedVerification - Автоматическая расширенная проверка сертификатов после завершении сессии. Допустимые значения: `true`, `false`.
- requireTLSv1 - Требование соединения TLSv1. Допустимые значения: `true`, `false`.
- requireTLSv1\_1 - Требование соединения TLSv1.1. Допустимые значения: `true`, `false`.
- requireTLSv1\_2 - Требование соединения TLSv1.2. Допустимые значения: `true`, `false`.
- fips - Активация режима OpenSSL FIPS. Поддерживается, если версия OpenSSL, с которой собрана библиотека поддерживает fips.
- privateKeyPassphraseHandler - Класс (подкласс `PrivateKeyPassphraseHandler`) запрашающий кодовую фразу доступа к секретному ключу. Например, `<privateKeyPassphraseHandler>`,  
`<name>KeyFileHandler</name>`, `<options><password>test</password></options>`,  
`</privateKeyPassphraseHandler>`.
- invalidCertificateHandler - Класс (подкласс `CertificateHandler`) для подтверждения не валидных сертификатов. Например, `<invalidCertificateHandler>` `<name>ConsoleCertificateHandler</name>`  
`</invalidCertificateHandler>`.
- disableProtocols - Запрещенные к использованию протоколы.
- preferServerCiphers - Предпочтение серверных шифров на клиенте.

#### **Пример настройки:**

```

<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

Логирование событий, связанных с данными типа [MergeTree](#). Например, события добавления или мержа данных. Лог можно использовать для симуляции алгоритмов слияния, чтобы сравнивать их характеристики. Также, можно визуализировать процесс слияния.

Запросы логируются не в отдельный файл, а в таблицу [system.part\\_log](#). Вы можете изменить название этой таблицы в параметре `table` (см. ниже).

При настройке логирования используются следующие параметры:

- `database` — имя базы данных;
- `table` — имя таблицы;
- `partition_by` — устанавливает [произвольный ключ партиционирования](#). Нельзя использовать если используется `engine`
- `engine` - устанавливает [настройки MergeTree Engine](#) для системной таблицы. Нельзя использовать если используется `partition_by`.
- `flush_interval_milliseconds` — период сброса данных из буфера в памяти в таблицу.

### Пример

```

<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

Путь к каталогу с данными.

### Обратите внимание

Завершающий слеш обязателен.

## Пример

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

## prometheus

Опубликовать данные о метриках, для сбора с помощью системы мониторинга [Prometheus](#).

Настройки:

- `endpoint` – путь по которому будет осуществляться экспорт данных метрик по HTTP протоколу для сбора с помощью `prometheus`. Должен начинаться с '/'.
- `port` – порт по которому будет доступен `endpoint` для сбора метрик.
- `metrics` – флаг для экспорта текущих значений метрик из таблицы [system.metrics](#).
- `events` – флаг для экспорта текущих значений метрик из таблицы [system.events](#).
- `asynchronous_metrics` – флаг для экспорта текущих значений значений метрик из таблицы [system.asynchronous\\_metrics](#).

## Пример

```
<prometheus>
  <endpoint>/metrics</endpoint>
  <port>8001</port>
  <metrics>true</metrics>
  <events>true</events>
  <asynchronous_metrics>true</asynchronous_metrics>
</prometheus>
```

## query\_log

Настройка логирования запросов, принятых с настройкой `log_queries=1`.

Запросы логируются не в отдельный файл, а в системную таблицу [system.query\\_log](#). Вы можете изменить название этой таблицы в параметре `table` (см. ниже).

При настройке логирования используются следующие параметры:

- `database` — имя базы данных;
- `table` — имя таблицы, куда будет записываться лог;
- `partition_by` — устанавливает [произвольный ключ партиционирования](#). Нельзя использовать если используется `engine`
- `engine` - устанавливает [настройки MergeTree Engine](#) для системной таблицы. Нельзя использовать если используется `partition_by`.
- `flush_interval_milliseconds` — период сброса данных из буфера в памяти в таблицу.

Если таблица не существует, то ClickHouse создаст её. Если структура журнала запросов изменилась при обновлении сервера ClickHouse, то таблица со старой структурой переименовывается, а новая таблица создается автоматически.

## Пример

```
<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

Настройка логирования потоков выполнения запросов, принятых с настройкой `log_query_threads=1`.

Запросы логируются не в отдельный файл, а в системную таблицу `system.query_thread_log`. Вы можете изменить название этой таблицы в параметре `table` (см. ниже).

При настройке логирования используются следующие параметры:

- `database` — имя базы данных;
- `table` — имя таблицы, куда будет записываться лог;
- `partition_by` — устанавливает [произвольный ключ партиционирования](#). Нельзя использовать если используется `engine`
- `engine` - устанавливает [настройки MergeTree Engine](#) для системной таблицы. Нельзя использовать если используется `partition_by`.
- `flush_interval_milliseconds` — период сброса данных из буфера в памяти в таблицу.

Если таблица не существует, то ClickHouse создаст её. Если структура журнала запросов изменилась при обновлении сервера ClickHouse, то таблица со старой структурой переименовывается, а новая таблица создается автоматически.

## Пример

```
<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

Настройки логирования информации о зависимых представлениях (materialized, live и т.п.) в запросах принятых с настройкой `log_query_views=1`.

Запросы сохраняются в таблицу `system.query_views_log`. Вы можете изменить название этой таблицы в параметре `table` (см. ниже).

При настройке логирования используются следующие параметры:

- `database` – имя базы данных.
- `table` – имя таблицы куда будут записываться использованные представления.
- `partition_by` — устанавливает [произвольный ключ партиционирования](#). Нельзя использовать если используется `engine`

- `engine` - устанавливает [настройки MergeTree Engine](#) для системной таблицы. Нельзя использовать если используется `partition_by`.
- `flush_interval_milliseconds` — период сброса данных из буфера в памяти в таблицу.

Если таблица не существует, то ClickHouse создаст её. Если структура журнала запросов изменилась при обновлении сервера ClickHouse, то таблица со старой структурой переименовывается, а новая таблица создается автоматически.

## 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

Настройка логирования текстовых сообщений в системную таблицу [text\\_log](#).

Параметры:

- `level` — Максимальный уровень сообщения (по умолчанию `Trace`) которое будет сохранено в таблице.
- `database` — имя базы данных для хранения таблицы.
- `table` — имя таблицы, куда будут записываться текстовые сообщения.
- `partition_by` — устанавливает [произвольный ключ партиционирования](#). Нельзя использовать если используется `engine`
- `engine` - устанавливает [настройки MergeTree Engine](#) для системной таблицы. Нельзя использовать если используется `partition_by`.
- `flush_interval_milliseconds` — период сброса данных из буфера в памяти в таблицу.

## Пример

```
<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

Настройки для [trace\\_log](#) system table operation.

Parameters:

- `database` — Database for storing a table.
- `table` — Table name.

- `partition_by` — устанавливает **произвольный ключ партиционирования**. Нельзя использовать если используется `engine`
- `engine` - устанавливает **настройки MergeTree Engine** для системной таблицы. Нельзя использовать если используется `partition_by`.
- `flush_interval_milliseconds` — Interval for flushing data from the buffer in memory to the table.

По умолчанию файл настроек сервера config.xml содержит следующие настройки:

```
<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

Правила основанные на регулярных выражениях, которые будут применены для всех запросов а также для всех сообщений перед сохранением их в лог на сервере, `system.query_log`, `system.text_log`, `system.processes` таблицы, а также в логах отсылаемых клиенту. Это позволяет предотвратить утечку конфиденциальных данных из SQL запросов (такие как имена, электронные письма, личные идентификаторы или номера кредитных карт) в логи.

### Пример

```
<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>
```

Параметры конфигурации:

- `name` - имя правила (необязательно)
- `regexp` - совместимое с RE2 регулярное выражение (обязательное)
- `replace` - строка замены для конфиденциальных данных (опционально, по умолчанию - шесть звездочек)

Правила маскировки применяются ко всему запросу (для предотвращения утечки конфиденциальных данных из неправильно оформленных / не интерпритируемых запросов).

`system.events` таблица содержит счетчик `QueryMaskingRulesMatch` который считает общее кол-во совпадений правил маскировки.

Для распределенных запросов каждый сервер должен быть сконфигурирован отдельно, иначе, подзапросы, переданные на другие узлы, будут сохраняться без маскировки.

## remote\_servers

Конфигурация кластеров, которые использует движок таблиц **Distributed** и табличная функция `cluster`.

### Пример

```
<remote_servers incl="clickhouse_remote_servers" />
```

Значение атрибута `incl` смотрите в разделе «[Конфигурационные файлы](#)».

### Смотрите также

- [skip\\_unavailable\\_shards](#)

## timezone

Временная зона сервера.

Указывается идентификатором IANA в виде часового пояса UTC или географического положения (например, Africa/Abidjan).

Временная зона необходима при преобразованиях между форматами `String` и `DateTime`, которые возникают при выводе полей `DateTime` в текстовый формат (на экран или в файл) и при получении `DateTime` из строки. Также, временная зона используется в функциях, которые работают со временем и датой, если они не получили временную зону в параметрах вызова.

### Пример

```
<timezone>Europe/Moscow</timezone>
```

## tcp\_port

Порт для взаимодействия с клиентами по протоколу TCP.

### Пример

```
<tcp_port>9000</tcp_port>
```

## tcp\_port\_secure

TCP порт для защищённого обмена данными с клиентами. Используйте с настройкой [OpenSSL](#).

### Возможные значения

Положительное целое число.

### Значение по умолчанию

```
<tcp_port_secure>9440</tcp_port_secure>
```

## mysql\_port

Порт для взаимодействия с клиентами по протоколу MySQL.

### Возможные значения

Положительное целое.

### Пример

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

## **tmp\_path**

Путь ко времененным данным для обработки больших запросов.

### Обратите внимание

Завершающий слеш обязателен.

#### Пример

```
<tmp_path>/var/lib/clickhouse/tmp/</tmp_path>
```

## **tmp\_policy**

Политика из [storage\\_configuration](#) для хранения временных файлов.

Если политика не задана, используется [tmp\\_path](#). В противном случае [tmp\\_path](#) игнорируется.

### Примечание

- [move\\_factor](#) игнорируется.
- [keep\\_free\\_space\\_bytes](#) игнорируется.
- [max\\_data\\_part\\_size\\_bytes](#) игнорируется.
- В данной политике у вас должен быть ровно один том.

## **uncompressed\_cache\_size**

Размер кеша (в байтах) для несжатых данных, используемых движками таблиц семейства [MergeTree](#).

Кеш единый для сервера. Память выделяется по требованию. Кеш используется в том случае, если включена опция [use\\_uncompressed\\_cache](#).

Несжатый кеш выгодно использовать для очень коротких запросов в отдельных случаях.

#### Пример

```
<uncompressed_cache_size>8589934592</uncompressed_cache_size>
```

## **user\_files\_path**

Каталог с пользовательскими файлами. Используется в табличной функции [file\(\)](#).

#### Пример

```
<user_files_path>/var/lib/clickhouse/user_files/</user_files_path>
```

## **users\_config**

Путь к файлу, который содержит:

- Конфигурации пользователей.
- Права доступа.
- Профили настроек.
- Настройки квот.

## Пример

```
<users_config>users.xml</users_config>
```

## zookeeper

Содержит параметры, позволяющие ClickHouse взаимодействовать с кластером ZooKeeper.

ClickHouse использует ZooKeeper для хранения метаданных о репликах при использовании реплицированных таблиц. Если реплицированные таблицы не используются, этот раздел параметров может отсутствовать.

Раздел содержит следующие параметры:

- `node` — адрес ноды (сервера) ZooKeeper. Можно сконфигурировать несколько нод.

Например:

```
<node index="1">
  <host>example_host</host>
  <port>2181</port>
</node>
```

Атрибут `index` задает порядок опроса нод при попытках подключиться к кластеру ZooKeeper.

- `session_timeout_ms` — максимальный таймаут клиентской сессии в миллисекундах.
- `operation_timeout_ms` — максимальный таймаут для одной операции в миллисекундах.
- `root` — `znode`, который используется как корневой для всех znode, которые использует сервер ClickHouse. Необязательный.
- `identity` — пользователь и пароль, которые может потребовать ZooKeeper для доступа к запрошенным znode. Необязательный.

## Пример конфигурации

```
<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>
```

## Смотрите также

- [Репликация](#)
- [ZooKeeper Programmer's Guide](#)

## use\_minimalistic\_part\_header\_in\_zookeeper

Способ хранения заголовков кусков данных в ZooKeeper.

Параметр применяется только к семейству таблиц `MergeTree`. Его можно установить:

- Глобально в разделе `merge_tree` файла `config.xml`.

ClickHouse использует этот параметр для всех таблиц на сервере. Вы можете изменить настройку в любое время. Существующие таблицы изменяют свое поведение при изменении параметра.

- Для каждой отдельной таблицы.

При создании таблицы укажите соответствующую [настройку движка](#). Поведение существующей таблицы с установленным параметром не изменяется даже при изменении глобального параметра.

### Возможные значения

- 0 — функциональность выключена.
- 1 — функциональность включена.

Если `use_minimalistic_part_header_in_zookeeper = 1`, то [реплицированные](#) таблицы хранят заголовки кусков данных в компактном виде, используя только одну `znode`. Если таблица содержит много столбцов, этот метод хранения значительно уменьшает объём данных, хранящихся в Zookeeper.

## Внимание

После того как вы установили `use_minimalistic_part_header_in_zookeeper = 1`, невозможно откатить ClickHouse до версии, которая не поддерживает этот параметр. Будьте осторожны при обновлении ClickHouse на серверах в кластере. Не обновляйте все серверы сразу. Безопаснее проверять новые версии ClickHouse в тестовой среде или только на некоторых серверах кластера.

Заголовки частей данных, ранее сохранённые с этим параметром, не могут быть восстановлены в их предыдущем (некомпактном) представлении.

**Значение по умолчанию:** 0.

## disable\_internal\_dns\_cache

Отключает внутренний кеш DNS записей. Используется при эксплуатации ClickHouse в системах с часто меняющейся инфраструктурой, таких как Kubernetes.

**Значение по умолчанию:** 0.

## dns\_cache\_update\_period

Период обновления IP адресов у записей во внутреннем DNS кеше ClickHouse (в секундах). Обновление выполняется асинхронно, отдельным системным потоком.

**Значение по умолчанию:** 15.

## **Смотрите также**

- [background\\_schedule\\_pool\\_size](#)

## **distributed\_ddl**

Управление запуском [распределенных ddl запросов](#) (CREATE, DROP, ALTER, RENAME) в кластере. Работает только если разрешена [работа с ZooKeeper](#).

### **Пример**

```
<distributed_ddl>
    <!-- Путь в ZooKeeper для очереди содержащей DDL запросы -->
    <path>/clickhouse/task_queue/ddl</path>

    <!-- Настройки из этого профиля будут использованы для запуска DDL запросов -->
    <profile>default</profile>

    <!-- Контроль того как много ON CLUSTER запросов могут исполняться одновременно. -->
    <pool_size>1</pool_size>

    <!--
        Настройки очистки (активные задачи в очереди не будут удаляться)
    -->

    <!-- Время TTL для задач в секундах (по умолчанию 1 week) -->
    <task_max_lifetime>604800</task_max_lifetime>

    <!-- Как часто будет запускаться очистка данных (в секундах) -->
    <cleanup_delay_period>60</cleanup_delay_period>

    <!-- Как много задач может быть в очереди -->
    <max_tasks_in_queue>1000</max_tasks_in_queue>
</distributed_ddl>
```

## **access\_control\_path**

Путь к каталогу, где сервер ClickHouse хранит конфигурации пользователей и ролей, созданные командами SQL.

Значение по умолчанию: `/var/lib/clickhouse/access/`.

## **Смотрите также**

- [Управление доступом](#)

## **user\_directories**

Секция конфигурационного файла, которая содержит настройки:

- Путь к конфигурационному файлу с предустановленными пользователями.
- Путь к файлу, в котором содержатся пользователи, созданные при помощи SQL команд.
- Путь к узлу ZooKeeper, где хранятся и реплицируются пользователи, созданные с помощью команд SQL (экспериментальная функциональность).

Если эта секция определена, путь из [users\\_config](#) и [access\\_control\\_path](#) не используется.

Секция `user_directories` может содержать любое количество элементов, порядок расположения элементов обозначает их приоритет (чем выше элемент, тем выше приоритет).

### **Примеры**

```
<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>
```

Пользователи, роли, политики доступа к строкам, квоты и профили могут храниться в ZooKeeper:

```
<user_directories>
  <users_xml>
    <path>/etc/clickhouse-server/users.xml</path>
  </users_xml>
  <replicated>
    <zookeeper_path>/clickhouse/access/</zookeeper_path>
  </replicated>
</user_directories>
```

Также вы можете добавить секции `memory` — означает хранение информации только в памяти, без записи на диск, и `ldap` — означает хранения информации на [LDAP-сервере](#).

Чтобы добавить LDAP-сервер в качестве удаленного каталога пользователей, которые не определены локально, определите один раздел `ldap` со следующими параметрами:

- `server` — имя одного из LDAP-серверов, определенных в секции `ldap_servers` конфигурационного файла. Этот параметр является необязательным и может быть пустым.
- `roles` — раздел со списком локально определенных ролей, которые будут назначены каждому пользователю, полученному с LDAP-сервера. Если роли не заданы, пользователь не сможет выполнять никаких действий после аутентификации. Если какая-либо из перечисленных ролей не определена локально во время проверки подлинности, попытка проверки подлинности завершится неудачей, как если бы предоставленный пароль был неверным.

## Пример

```
<ldap>
  <server>my_ldap_server</server>
  <roles>
    <my_local_role1 />
    <my_local_role2 />
  </roles>
</ldap>
```

# Настройки

Все настройки, описанные ниже, могут быть заданы несколькими способами.

Настройки задаются послойно, т.е. каждый следующий слой перезаписывает предыдущие настройки.

Способы задания настроек, упорядоченные по приоритету:

- Настройки в конфигурационном файле сервера `users.xml`.

Устанавливаются в элементе `<profiles>`.

- Настройки для сессии.

Из консольного клиента ClickHouse в интерактивном режиме отправьте запрос `SET setting=value`. Аналогично можно использовать ClickHouse-сессии в HTTP-протоколе, для этого необходимо указывать HTTP-параметр `session_id`.

- Настройки для запроса.

- При запуске консольного клиента ClickHouse в не интерактивном режиме установите параметр запуска `--setting=value`.
- При использовании HTTP API передавайте cgi-параметры (`URL?setting_1=value&setting_2=value...`).
- Укажите необходимые настройки в секции **SETTINGS** запроса `SELECT`. Эти настройки действуют только в рамках данного запроса, а после его выполнения сбрасываются до предыдущего значения или значения по умолчанию.

Настройки, которые можно задать только в конфигурационном файле сервера, в разделе не рассматриваются.

## Пользовательские настройки

В дополнение к общим [настройкам](#), пользователи могут определять собственные настройки.

Название пользовательской настройки должно начинаться с одного из предопределённых префиксов. Список этих префиксов должен быть задан в параметре `custom_settings_prefixes` конфигурационного файла сервера.

```
<custom_settings_prefixes>custom_</custom_settings_prefixes>
```

Чтобы задать значение пользовательской настройке, используйте команду `SET`:

```
SET custom_a = 123;
```

Чтобы получить текущее значение пользовательской настройки, используйте функцию `getSetting()`:

```
SELECT getSetting('custom_a');
```

### См. также

- [Конфигурационные параметры сервера](#)

## Разрешения для запросов

Запросы в ClickHouse можно разделить на несколько типов:

1. Запросы на чтение данных: `SELECT`, `SHOW`, `DESCRIBE`, `EXISTS`.
2. Запросы за запись данных: `INSERT`, `OPTIMIZE`.
3. Запросы на изменение настроек: `SET`, `USE`.
4. [Запросы DDL](#): `CREATE`, `ALTER`, `RENAME`, `ATTACH`, `DETACH`, `DROP`, `TRUNCATE`.
5. `KILL QUERY`.

Разрешения пользователя по типу запроса регулируются параметрами:

- `readonly` — ограничивает разрешения для всех типов запросов, кроме DDL.

- **allow\_ddl** — ограничивает разрешения для DDL запросов.

KILL QUERY выполняется с любыми настройками.

## readonly

Ограничивает разрешения для запросов на чтение данных, запись данных и изменение параметров.

Разделение запросов по типам смотрите по тексту [выше](#) по тексту.

### Возможные значения

- 0 — разрешены все запросы.
- 1 — разрешены только запросы на чтение данных.
- 2 — разрешены запросы на чтение данных и изменение настроек.

После установки `readonly = 1` или `2` пользователь не может изменить настройки `readonly` и `allow_ddl` в текущей сессии.

При использовании метода GET в [HTTP интерфейсе](#), `readonly = 1` устанавливается автоматически. Для изменения данных используйте метод `POST`.

Установка `readonly = 1` запрещает изменение всех настроек. Существует способ запретить изменения только некоторых настроек, см. [ограничения на изменение настроек](#).

### Значение по умолчанию

0

## allow\_ddl

Разрешает/запрещает [DDL](#) запросы.

Разделение запросов по типам смотрите по тексту [выше](#) по тексту.

### Возможные значения

- 0 — DDL запросы не разрешены.
- 1 — DDL запросы разрешены.

Если `allow_ddl = 0`, то невозможно выполнить `SET allow_ddl = 1` для текущей сессии.

### Значение по умолчанию

1

## Ограничения на сложность запроса

Ограничения на сложность запроса - часть настроек.

Используются, чтобы обеспечить более безопасное исполнение запросов из пользовательского интерфейса.

Почти все ограничения действуют только на SELECT-ы.

При распределённой обработке запроса, ограничения действуют на каждом сервере по отдельности.

Ограничения проверяются на каждый блок обработанных данных, а не на каждую строку. В связи с этим, ограничения могут быть превышены на размер блока.

Ограничения вида «максимальное количество чего-нибудь» могут принимать значение 0, которое обозначает «не ограничено».

Для большинства ограничений также присутствует настройка вида `overflow_mode` - что делать, когда ограничение превышено.

Оно может принимать одно из двух значений: `throw` или `break`; а для ограничения на агрегацию (`group_by_overflow_mode`) есть ещё значение `any`.

`throw` - кинуть исключение (по умолчанию).

`break` - прервать выполнение запроса и вернуть неполный результат, как будто исходные данные закончились.

`any` (только для `group_by_overflow_mode`) - продолжить агрегацию по ключам, которые успели войти в набор, но не добавлять новые ключи в набор.

## `max_memory_usage`

Максимальный возможный объём оперативной памяти для выполнения запроса на одном сервере.

В конфигурационном файле по умолчанию, ограничение равно 10 ГБ.

Настройка не учитывает объём свободной памяти или общий объём памяти на машине.

Ограничение действует на один запрос, в пределах одного сервера.

Текущее потребление памяти для каждого запроса можно посмотреть с помощью `SHOW PROCESSLIST`. Также отслеживается и выводится в лог пиковое потребление памяти для каждого запроса.

Потребление памяти не отслеживается для состояний некоторых агрегатных функций.

Потребление памяти не полностью учитывается для состояний агрегатных функций `min`, `max`, `any`, `anyLast`, `argMin`, `argMax` от аргументов `String` и `Array`.

Потребление памяти ограничивается также параметрами `max_memory_usage_for_user` и `max_server_memory_usage`.

## `max_memory_usage_for_user`

Максимальный возможный объём оперативной памяти для запросов пользователя на одном сервере.

Значения по умолчанию определены в файле `Settings.h`. По умолчанию размер не ограничен (`max_memory_usage_for_user = 0`).

Смотрите также описание настройки `max_memory_usage`.

## `max_rows_to_read`

Следующие ограничения могут проверяться на каждый блок (а не на каждую строку). То есть, ограничения могут быть немного нарушены.

Максимальное количество строчек, которое можно прочитать из таблицы при выполнении запроса.

## `max_bytes_to_read`

Максимальное количество байт (несжатых данных), которое можно прочитать из таблицы при выполнении запроса.

## `read_overflow_mode`

Что делать, когда количество прочитанных данных превысило одно из ограничений: `throw` или `break`. По умолчанию: `throw`.

## `max_rows_to_read_leaf`

Следующие ограничения могут проверяться на каждый блок (а не на каждую строку). То есть, ограничения могут быть немного нарушены.

Максимальное количество строчек, которое можно прочитать из таблицы на удалённом сервере при выполнении распределенного запроса. Распределенные запросы могут создавать несколько подзапросов к каждому из шардов в кластере и тогда этот лимит будет применен при выполнении чтения на удаленных серверах (включая и сервер-инициатор) и проигнорирован на сервере-инициаторе запроса во время объединения полученных результатов. Например, кластер состоит из 2 шардов и каждый из них хранит таблицу с 100 строками. Тогда распределенный запрос для получения всех данных из этих таблиц и установленной настройкой `max_rows_to_read=150` выбросит исключение, т.к. в общем он прочитает 200 строк. Но запрос с настройкой `max_rows_to_read_leaf=150` завершится успешно, потому что каждый из шардов прочитает максимум 100 строк.

## `max_bytes_to_read_leaf`

Максимальное количество байт (несжатых данных), которое можно прочитать из таблицы на удалённом сервере при выполнении распределенного запроса. Распределенные запросы могут создавать несколько подзапросов к каждому из шардов в кластере и тогда этот лимит будет применен при выполнении чтения на удаленных серверах (включая и сервер-инициатор) и проигнорирован на сервере-инициаторе запроса во время объединения полученных результатов. Например, кластер состоит из 2 шардов и каждый из них хранит таблицу со 100 байтами. Тогда распределенный запрос для получения всех данных из этих таблиц и установленной настройкой `max_bytes_to_read=150` выбросит исключение, т.к. в общем он прочитает 200 байт. Но запрос с настройкой `max_bytes_to_read_leaf=150` завершится успешно, потому что каждый из шардов прочитает максимум 100 байт.

## `read_overflow_mode_leaf`

Что делать, когда количество прочитанных данных на удаленном сервере превысило одно из ограничений: `throw` или `break`. По умолчанию: `throw`.

## `max_rows_to_group_by`

Максимальное количество уникальных ключей, получаемых в процессе агрегации. Позволяет ограничить потребление оперативки при агрегации.

## `group_by_overflow_mode`

Что делать, когда количество уникальных ключей при агрегации превысило ограничение: `throw`, `break` или `any`. По умолчанию: `throw`.

Использование значения `any` позволяет выполнить GROUP BY приближенно. Качество такого приближенного вычисления сильно зависит от статистических свойств данных.

## `max_bytes_before_external_group_by`

Включает или отключает выполнение секций GROUP BY во внешней памяти. Смотрите [GROUP BY во внешней памяти](#).

Возможные значения:

- Максимальный объём RAM (в байтах), который может использовать отдельная операция GROUP BY.
- 0 — GROUP BY во внешней памяти отключен.

Значение по умолчанию — 0.

## max\_rows\_to\_sort

Максимальное количество строк до сортировки. Позволяет ограничить потребление оперативки при сортировке.

## max\_bytes\_to\_sort

Максимальное количество байт до сортировки.

## sort\_overflow\_mode

Что делать, если количество строк, полученное перед сортировкой, превысило одно из ограничений: throw или break. По умолчанию: throw.

## max\_result\_rows

Ограничение на количество строк результата. Проверяются также для подзапросов и на удалённых серверах при выполнении части распределённого запроса.

## max\_result\_bytes

Ограничение на количество байт результата. Аналогично.

## result\_overflow\_mode

Что делать, если объём результата превысил одно из ограничений: throw или break. По умолчанию: throw.

Использование break по смыслу похоже на LIMIT. Break прерывает выполнение только на уровне блока. Т.е. число строк которые вернет запрос будет больше чем ограничение [max\\_result\\_rows](#), кратно [max\\_block\\_size](#) и зависит от [max\\_threads](#).

Пример:

```
SET max_threads = 3, max_block_size = 3333;
SET max_result_rows = 3334, result_overflow_mode = 'break';

SELECT *
FROM numbers_mt(100000)
FORMAT Null;
```

Результат:

```
6666 rows in set. ...
```

## max\_execution\_time

Максимальное время выполнения запроса в секундах.

На данный момент не проверяется при одной из стадий сортировки а также при слиянии и финализации агрегатных функций.

## timeout\_overflow\_mode

Что делать, если запрос выполняется дольше max\_execution\_time: throw или break. По умолчанию: throw.

## min\_execution\_speed

Минимальная скорость выполнения запроса в строчках в секунду. Проверяется на каждый блок данных по истечении timeout\_before\_checking\_execution\_speed. Если скорость выполнения запроса оказывается меньше, то кидается исключение.

## min\_execution\_speed\_bytes

Минимальная скорость выполнения запроса в строках на байт. Он проверяется для каждого блока данных после timeout\_before\_checking\_execution\_speed. Если скорость выполнения запроса меньше, исключение.

## max\_execution\_speed

Максимальная скорость выполнения запроса в строках в секунду. Он проверяется для каждого блока данных после timeout\_before\_checking\_execution\_speed. Если скорость выполнения запроса выше, скорость будет снижена.

## max\_execution\_speed\_bytes

Максимальная скорость выполнения запроса в байтах в секунду. Он проверяется для каждого блока данных после timeout\_before\_checking\_execution\_speed. Если скорость выполнения запроса выше, скорость будет снижена.

## timeout\_before\_checking\_execution\_speed

Проверять, что скорость выполнения запроса не слишком низкая (не меньше min\_execution\_speed), после прошествия указанного времени в секундах.

## max\_columns\_to\_read

Максимальное количество столбцов, которых можно читать из таблицы в одном запросе. Если запрос требует чтения большего количества столбцов - кинуть исключение.

## max\_temporary\_columns

Максимальное количество временных столбцов, которых необходимо одновременно держать в оперативке, в процессе выполнения запроса, включая константные столбцы. Если временных столбцов оказалось больше - кидается исключение.

## max\_temporary\_non\_const\_columns

То же самое, что и max\_temporary\_columns, но без учёта столбцов-констант.

Стоит заметить, что столбцы-константы довольно часто образуются в процессе выполнения запроса, но расходуют примерно нулевое количество вычислительных ресурсов.

## max\_subquery\_depth

Максимальная вложенность подзапросов. Если подзапросы более глубокие - кидается исключение. По умолчанию: 100.

## max\_pipeline\_depth

Максимальная глубина конвейера выполнения запроса. Соответствует количеству преобразований, которое проходит каждый блок данных в процессе выполнения запроса. Считается в пределах одного сервера. Если глубина конвейера больше - кидается исключение. По умолчанию: 1000.

## max\_ast\_depth

Максимальная вложенность синтаксического дерева запроса. Если превышена - кидается исключение.

На данный момент, проверяются не во время парсинга а уже после парсинга запроса. То есть, во время парсинга может быть создано слишком глубокое синтаксическое дерево, но запрос не будет выполнен. По умолчанию: 1000.

## max\_ast\_elements

Максимальное количество элементов синтаксического дерева запроса. Если превышено - кидается исключение.

Аналогично, проверяется уже после парсинга запроса. По умолчанию: 50 000.

## max\_rows\_in\_set

Максимальное количество строчек для множества в секции IN, создаваемого из подзапроса.

## max\_bytes\_in\_set

Максимальное количество байт (ненжатых данных), занимаемое множеством в секции IN, создаваемым из подзапроса.

## set\_overflow\_mode

Что делать, когда количество данных превысило одно из ограничений: throw или break. По умолчанию: throw.

## max\_rows\_in\_distinct

Максимальное количество различных строчек при использовании DISTINCT.

## max\_bytes\_in\_distinct

Максимальное количество байт, занимаемых хэш-таблицей, при использовании DISTINCT.

## distinct\_overflow\_mode

Что делать, когда количество данных превысило одно из ограничений: throw или break. По умолчанию: throw.

## max\_rows\_to\_transfer

Максимальное количество строчек, которых можно передать на удалённый сервер или сохранить во временную таблицу, при использовании GLOBAL IN.

## max\_bytes\_to\_transfer

Максимальное количество байт (несжатых данных), которых можно передать на удалённый сервер или сохранить во временную таблицу, при использовании GLOBAL IN.

## transfer\_overflow\_mode

Что делать, когда количество данных превысило одно из ограничений: throw или break. По умолчанию: throw.

## max\_rows\_in\_join

Ограничивает количество строк в хэш-таблице, используемой при соединении таблиц.

Параметр применяется к операциям `SELECT... JOIN` и к движку таблиц `Join`.

Если запрос содержит несколько `JOIN`, то ClickHouse проверяет значение настройки для каждого промежуточного результата.

При достижении предела ClickHouse может выполнять различные действия. Используйте настройку `join_overflow_mode` для выбора действия.

Возможные значения:

- Положительное целое число.
- 0 — неограниченное количество строк.

Значение по умолчанию — 0.

## max\_bytes\_in\_join

Ограничивает размер (в байтах) хэш-таблицы, используемой при объединении таблиц.

Параметр применяется к операциям `SELECT... JOIN` и к движку таблиц `Join`.

Если запрос содержит несколько `JOIN`, то ClickHouse проверяет значение настройки для каждого промежуточного результата.

При достижении предела ClickHouse может выполнять различные действия. Используйте настройку `join_overflow_mode` для выбора действия.

Возможные значения:

- Положительное целое число.
- 0 — контроль памяти отключен.

Значение по умолчанию — 0.

## join\_overflow\_mode

Определяет, какое действие ClickHouse выполняет при достижении любого из следующих ограничений для `JOIN`:

- `max_bytes_in_join`
- `max_rows_in_join`

Возможные значения:

- `THROW` — ClickHouse генерирует исключение и прерывает операцию.
- `BREAK` — ClickHouse прерывает операцию, но не генерирует исключение.

Значение по умолчанию — `THROW`.

## Смотрите также

- [Секция JOIN](#)
- [Движок таблиц Join](#)

## max\_partitions\_per\_insert\_block

Ограничивает максимальное количествоパーティций в одном вставленном блоке.

- Положительное целое число.
- 0 — неограниченное количество разделов.

Значение по умолчанию: 100.

## Подробности

При вставке данных, ClickHouse вычисляет количествоパーティций во вставленном блоке. Если числоパーティций больше, чем `max_partitions_per_insert_block`, ClickHouse генерирует исключение со следующим текстом:

```
<Too many partitions for single INSERT block (more than) + toString(max_parts) + >. The limit is controlled by 'max_partitions_per_insert_block' setting. 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.).>
```

# Настройки

## distributed\_product\_mode

Изменяет поведение [распределенных подзапросов](#).

ClickHouse применяет настройку в тех случаях, когда запрос содержит произведение распределённых таблиц, т.е. когда запрос к распределенной таблице содержит не-GLOBAL подзапрос к также распределенной таблице.

Условия применения:

- Только подзапросы для `IN`, `JOIN`.
- Только если в секции `FROM` используется распределённая таблица, содержащая более одного шарда.
- Если подзапрос касается распределенной таблицы, содержащей более одного шарда.
- Не используется в случае табличной функции `remote`.

Возможные значения:

- `deny` — значение по умолчанию. Запрещает использование таких подзапросов (При попытке использования вернет исключение «Double-distributed IN/JOIN subqueries is denied»);

- `local` — заменяет базу данных и таблицу в подзапросе на локальные для конечного сервера (шарда), оставив обычный `IN/JOIN`.
- `global` — заменяет запрос `IN/JOIN` на `GLOBAL IN/GLOBAL JOIN`.
- `allow` — разрешает использование таких подзапросов.

## `prefer_global_in_and_join`

Заменяет запрос `IN/JOIN` на `GLOBAL IN/GLOBAL JOIN`.

Возможные значения:

- 0 — выключена. Операторы `IN/JOIN` не заменяются на `GLOBAL IN/GLOBAL JOIN`.
- 1 — включена. Операторы `IN/JOIN` заменяются на `GLOBAL IN/GLOBAL JOIN`.

Значение по умолчанию: 0.

### Использование

Настройка `SET distributed_product_mode=global` меняет поведение запросов для распределенных таблиц, но она не подходит для локальных таблиц или таблиц из внешних источников. В этих случаях удобно использовать настройку `prefer_global_in_and_join`.

Например, если нужно объединить все данные из локальных таблиц, которые находятся на разных узлах — для распределенной обработки необходим `GLOBAL JOIN`.

Другой вариант использования настройки `prefer_global_in_and_join` — регулирование обращений к таблицам из внешних источников.

Эта настройка помогает уменьшить количество обращений к внешним ресурсам при объединении внешних таблиц: только один вызов на весь распределенный запрос.

### См. также:

- [Распределенные подзапросы `GLOBAL IN/GLOBAL JOIN`](#)

## `enable_optimize_predicate_expression`

Включает прорасывание предикатов в подзапросы для запросов `SELECT`.

Прорасывание предикатов может существенно уменьшить сетевой трафик для распределенных запросов.

Возможные значения:

- 0 — выключена.
- 1 — включена.

Значение по умолчанию: 1.

### Использование

Рассмотрим следующие запросы:

1. `SELECT count() FROM test_table WHERE date = '2018-10-10'`
2. `SELECT count() FROM (SELECT * FROM test_table) WHERE date = '2018-10-10'`

Если `enable_optimize_predicate_expression = 1`, то время выполнения запросов одинаковое, так как ClickHouse применяет `WHERE` к подзапросу сразу при его обработке.

Если `enable_optimize_predicate_expression = 0`, то время выполнения второго запроса намного больше, потому что секция `WHERE` применяется к данным уже после завершения подзапроса.

## `fallback_to_stale_replicas_for_distributed_queries`

Форсирует запрос в устаревшую реплику в случае, если актуальные данные недоступны. См. [Репликация](#).

Из устаревших реплик таблицы ClickHouse выбирает наиболее актуальную.

Используется при выполнении `SELECT` из распределенной таблицы, которая указывает на реплицированные таблицы.

По умолчанию - 1 (включена).

## `force_index_by_date`

Запрещает выполнение запросов, если использовать индекс по дате невозможно.

Работает с таблицами семейства MergeTree.

При `force_index_by_date=1` ClickHouse проверяет, есть ли в запросе условие на ключ даты, которое может использоваться для отсечения диапазонов данных. Если подходящего условия нет - кидается исключение. При этом не проверяется, действительно ли условие уменьшает объём данных для чтения. Например, условие `Date != '2000-01-01'` подходит даже в том случае, когда соответствует всем данным в таблице (т.е. для выполнения запроса требуется full scan). Подробнее про диапазоны данных в таблицах MergeTree читайте в разделе [MergeTree](#).

## `force_primary_key`

Запрещает выполнение запросов, если использовать индекс по первичному ключу невозможно.

Работает с таблицами семейства MergeTree.

При `force_primary_key=1` ClickHouse проверяет, есть ли в запросе условие на первичный ключ, которое может использоваться для отсечения диапазонов данных. Если подходящего условия нет - кидается исключение. При этом не проверяется, действительно ли условие уменьшает объём данных для чтения. Подробнее про диапазоны данных в таблицах MergeTree читайте в разделе [MergeTree](#).

## `format_schema`

Параметр применяется в том случае, когда используются форматы, требующие определения схемы, например [Cap'n Proto](#) или [Protobuf](#). Значение параметра зависит от формата.

## `fsync_metadata`

Включает или отключает `fsync` при записи `.sql` файлов. По умолчанию включено.

Имеет смысл выключать, если на сервере миллионы мелких таблиц-чанков, которые постоянно создаются и уничтожаются.

## `function_range_max_elements_in_block`

Устанавливает порог безопасности для объема данных, создаваемого функцией `range`. Задаёт максимальное количество значений, генерируемых функцией на блок данных (сумма размеров массивов для каждой строки в блоке).

Возможные значения:

- Положительное целое.

Значение по умолчанию: 500 000 000.

#### **См. также**

- [max\\_block\\_size](#)
- [min\\_insert\\_block\\_size\\_rows](#)

## **enable\_http\_compression**

Включает или отключает сжатие данных в ответе на HTTP-запрос.

Для получения дополнительной информации, читайте [Описание интерфейса HTTP](#).

Возможные значения:

- 0 — выключена.
- 1 — включена.

Значение по умолчанию: 0.

## **http\_zlib\_compression\_level**

Задаёт уровень сжатия данных в ответе на HTTP-запрос, если [enable\\_http\\_compression = 1](#).

Возможные значения: числа от 1 до 9.

Значение по умолчанию: 3.

## **http\_native\_compression\_disable\_checksumming\_on\_decompression**

Включает или отключает проверку контрольной суммы при распаковке данных HTTP POST от клиента. Используется только для собственного (Navite) формата сжатия ClickHouse (ни `gzip`, ни `deflate`).

Для получения дополнительной информации, читайте [Описание интерфейса HTTP](#).

Возможные значения:

- 0 — выключена.
- 1 — включена.

Значение по умолчанию: 0.

## **http\_max\_uri\_size**

Устанавливает максимальную длину URI в HTTP-запросе.

Возможные значения:

- Положительное целое.

Значение по умолчанию: 1048576.

## **table\_function\_remote\_max\_addresses**

Задает максимальное количество адресов, которые могут быть сгенерированы из шаблонов для функции `remote`.

Возможные значения:

- Положительное целое.

Значение по умолчанию: 1000.

## glob\_expansion\_max\_elements

Задает максимальное количество адресов, которые могут быть сгенерированы из шаблонов при использовании внешних хранилищ и при вызове табличных функциях (например, `url`), кроме функции `remote`.

Возможные значения:

- Положительное целое.

Значение по умолчанию: 1000.

## send\_progress\_in\_http\_headers

Включает или отключает HTTP-заголовки X-ClickHouse-Progress в ответах clickhouse-server.

Для получения дополнительной информации, читайте [Описание интерфейса HTTP](#).

Возможные значения:

- 0 — выключена.
- 1 — включена.

Значение по умолчанию: 0.

## max\_http\_get\_redirects

Ограничивает максимальное количество переходов по редиректам в таблицах с движком `URL` при выполнении HTTP запросов методом GET. Настройка применяется для обоих типов таблиц: созданных запросом `CREATE TABLE` и с помощью табличной функции `url`.

Возможные значения:

- Положительное целое число переходов.
- 0 — переходы запрещены.

Значение по умолчанию: 0.

## input\_format\_allow\_errors\_num

Устанавливает максимальное количество допустимых ошибок при чтении из текстовых форматов (CSV, TSV и т.п.).

Значение по умолчанию: 0.

Используйте обязательно в паре с `input_format_allow_errors_ratio`. Для пропуска ошибок, значения обеих настроек должны быть больше 0.

Если при чтении строки возникла ошибка, но при этом счетчик ошибок меньше `input_format_allow_errors_num`, то ClickHouse игнорирует строку и переходит к следующей.

В случае превышения `input_format_allow_errors_num` ClickHouse генерирует исключение.

## input\_format\_allow\_errors\_ratio

Устанавливает максимальную долю допустимых ошибок при чтении из текстовых форматов (CSV, TSV и т.п.).

Доля ошибок задаётся в виде числа с плавающей запятой от 0 до 1.

Значение по умолчанию: 0.

Используйте обязательно в паре с `input_format_allow_errors_num`. Для пропуска ошибок, значения обеих настроек должны быть больше 0.

Если при чтении строки возникла ошибка, но при этом текущая доля ошибок меньше `input_format_allow_errors_ratio`, то ClickHouse игнорирует строку и переходит к следующей.

В случае превышения `input_format_allow_errors_ratio` ClickHouse генерирует исключение.

## `input_format_parquet_import_nested`

Включает или отключает возможность вставки данных в колонки типа **Nested** в виде массива структур в формате ввода **Parquet**.

Возможные значения:

- 0 — данные не могут быть вставлены в колонки типа **Nested** в виде массива структур.
- 0 — данные могут быть вставлены в колонки типа **Nested** в виде массива структур.

Значение по умолчанию: 0.

## `input_format_arrow_import_nested`

Включает или отключает возможность вставки данных в колонки типа **Nested** в виде массива структур в формате ввода **Arrow**.

Возможные значения:

- 0 — данные не могут быть вставлены в колонки типа **Nested** в виде массива структур.
- 0 — данные могут быть вставлены в колонки типа **Nested** в виде массива структур.

Значение по умолчанию: 0.

## `input_format_orc_import_nested`

Включает или отключает возможность вставки данных в колонки типа **Nested** в виде массива структур в формате ввода **ORC**.

Возможные значения:

- 0 — данные не могут быть вставлены в колонки типа **Nested** в виде массива структур.
- 0 — данные могут быть вставлены в колонки типа **Nested** в виде массива структур.

Значение по умолчанию: 0.

## `input_format_values_interpret_expressions`

Включает или отключает парсер SQL, если потоковый парсер не может проанализировать данные. Этот параметр используется только для формата **Values** при вставке данных. Дополнительные сведения о парсерах читайте в разделе [Синтаксис](#).

Возможные значения:

- 0 — выключена.

В этом случае необходимо вставлять форматированные данные. Смотрите раздел [Форматы](#interfaces-formats-md).

- 1 — включена.

В этом случае вы можете использовать выражение SQL в качестве значения, но вставка данных намного медленнее. Если вы вставляете только форматированные данные, ClickHouse ведет себя так, как будто значение параметра равно 0.

Значение по умолчанию: 1.

Пример использования:

Вставим значение типа **DateTime** при разных значения настройки.

```
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.

Последний запрос эквивалентен следующему:

```
SET input_format_values_interpret_expressions = 0;
INSERT INTO datetime_t SELECT now()
```

Ok.

## input\_format\_values\_deduce\_templates\_of\_expressions

Включает или отключает попытку вычисления шаблона для выражений SQL в формате **Values**. Это позволяет гораздо быстрее парсить и интерпретировать выражения в **Values**, если выражения в последовательных строках имеют одинаковую структуру. ClickHouse пытается вычислить шаблон выражения, распарсить следующие строки с помощью этого шаблона и вычислить выражение в пачке успешно проанализированных строк.

Возможные значения:

- 0 — Выключена.
- 1 — Включена.

Значение по умолчанию: 1.

Для следующего запроса:

```
INSERT INTO test VALUES (lower('Hello')), (lower('world')), (lower('INSERT')), (upper("Values")), ...
```

- Если `input_format_values_interpret_expressions=1` и `format_values_deduce_templates_of_expressions=0`, выражения интерпретируются отдельно для каждой строки (это очень медленно для большого количества строк).
- Если `input_format_values_interpret_expressions=0` и `format_values_deduce_templates_of_expressions=1`, выражения в первой, второй и третьей строках парсятся с помощью шаблона `lower(String)` и интерпретируются вместе, выражение в четвертой строке парсится с другим шаблоном (`upper(String)`).
- Если `input_format_values_interpret_expressions=1` и `format_values_deduce_templates_of_expressions=1`, то же самое, что и в предыдущем случае, но также позволяет выполнять резервную интерпретацию выражений отдельно, если невозможно вычислить шаблон.

## input\_format\_values\_accurate\_types\_of\_literals

Эта настройка используется, только когда `input_format_values_deduce_templates_of_expressions = 1`. Выражения для некоторых столбцов могут иметь одинаковую структуру, но содержат числовые литералы разных типов, например:

```
(..., abs(0), ...),      -- UInt64 literal
(..., abs(3.141592654), ...), -- Float64 literal
(..., abs(-1), ...),     -- Int64 literal
```

Возможные значения:

- 0 — Выключена.

В этом случае, ClickHouse может использовать более общий тип для некоторых литералов (например, `Float64` или `Int64` вместо `UInt64` для 42), но это может привести к переполнению и проблемам с точностью.

- 1 — Включена.

В этом случае, ClickHouse проверяет фактический тип литерала и использует шаблон выражения соответствующего типа. В некоторых случаях это может значительно замедлить оценку выражения в `Values`.

Значение по умолчанию: 1.

## input\_format\_defaults\_for\_omitted\_fields

При вставке данных запросом `INSERT`, заменяет пропущенные поля значениям по умолчанию для типа данных столбца.

Поддерживаемые форматы вставки:

- `JSONEachRow`
- `CSV`
- `TabSeparated`

### Примечание

Когда опция включена, сервер отправляет клиенту расширенные метаданные. Это требует дополнительных вычислительных ресурсов на сервере и может снизить производительность.

Возможные значения:

- 0 — выключена.
- 1 — включена.

Значение по умолчанию: 1.

## input\_format\_tsv\_empty\_as\_default

Если эта настройка включена, все пустые поля во входящем TSV заменяются значениями по умолчанию. Для сложных выражений по умолчанию также должна быть включена настройка `input_format_defaults_for_omitted_fields`.

По умолчанию отключена.

## input\_format\_tsv\_enum\_as\_number

Включает или отключает парсинг значений перечислений как идентификаторов перечислений для входного формата TSV.

Возможные значения:

- 0 — парсинг значений перечисления как значений.
- 1 — парсинг значений перечисления как идентификаторов перечисления.

Значение по умолчанию: 0.

### Пример

Рассмотрим таблицу:

```
CREATE TABLE table_with_enum_column_for_tsv_insert (Id Int32, Value Enum('first' = 1, 'second' = 2))
ENGINE=Memory();
```

При включенной настройке `input_format_tsv_enum_as_number`:

```
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;
```

Результат:

Id	Value
102	second
103	first

При отключенной настройке `input_format_tsv_enum_as_number` запрос `INSERT`:

```
SET input_format_tsv_enum_as_number = 0;
INSERT INTO table_with_enum_column_for_tsv_insert FORMAT TSV 102 2;
```

сгенерирует исключение.

## input\_format\_null\_as\_default

Включает или отключает инициализацию **значениями по умолчанию** ячеек с **NULL**, если тип данных столбца не позволяет **хранить NULL**.

Если столбец не позволяет хранить **NULL** и эта настройка отключена, то вставка **NULL** приведет к возникновению исключения. Если столбец позволяет хранить **NULL**, то значения **NULL** вставляются независимо от этой настройки.

Эта настройка используется для запросов **INSERT ... VALUES** для текстовых входных форматов.

Возможные значения:

- 0 — вставка **NULL** в столбец, не позволяющий хранить **NULL**, приведет к возникновению исключения.
- 1 — ячейки с **NULL** инициализируются значением столбца по умолчанию.

Значение по умолчанию: 1.

## insert\_null\_as\_default

Включает или отключает вставку **значений по умолчанию** вместо **NULL** в столбцы, которые не позволяют **хранить NULL**.

Если столбец не позволяет хранить **NULL** и эта настройка отключена, то вставка **NULL** приведет к возникновению исключения. Если столбец позволяет хранить **NULL**, то значения **NULL** вставляются независимо от этой настройки.

Эта настройка используется для запросов **INSERT ... SELECT**. При этом подзапросы **SELECT** могут объединяться с помощью **UNION ALL**.

Возможные значения:

- 0 — вставка **NULL** в столбец, не позволяющий хранить **NULL**, приведет к возникновению исключения.
- 1 — вместо **NULL** вставляется значение столбца по умолчанию.

Значение по умолчанию: 1.

## input\_format\_skip\_unknown\_fields

Включает или отключает пропускание вставки неизвестных данных.

При записи данных, если входные данные содержат столбцы, которых нет в целевой таблице, ClickHouse генерирует исключение. Если пропускание вставки включено, ClickHouse не вставляет неизвестные данные и не генерирует исключение.

Поддерживаемые форматы:

- [JSONEachRow](#)
- [CSVWithNames](#)
- [TabSeparatedWithNames](#)
- [TSKV](#)

Возможные значения:

- 0 — выключена.
- 1 — включена.

Значение по умолчанию: 0.

## input\_format\_import\_nested\_json

Включает или отключает вставку данных JSON с вложенными объектами.

Поддерживаемые форматы:

- [JSONEachRow](#)

Возможные значения:

- 0 — выключена.
- 1 — включена.

Значение по умолчанию: 0.

См. также:

- [Использование вложенных структур](#) with the `JSONEachRow` format.

## input\_format\_with\_names\_use\_header

Включает или отключает проверку порядка столбцов при вставке данных.

Чтобы повысить эффективность вставки данных, рекомендуем отключить эту проверку, если вы уверены, что порядок столбцов входных данных такой же, как в целевой таблице.

Поддерживаемые форматы:

- [CSVWithNames](#)
- [TabSeparatedWithNames](#)

Возможные значения:

- 0 — выключена.
- 1 — включена.

Значение по умолчанию: 1.

## date\_time\_input\_format

Выбор парсера для текстового представления дат и времени при обработке входного формата.

Настройка не применяется к [функциям для работы с датой и временем](#).

Возможные значения:

- `best_effort` — включает расширенный парсинг.

ClickHouse может парсить базовый формат `YYYY-MM-DD HH:MM:SS` и все форматы [ISO 8601](#). Например, `2018-06-08T01:02:03.000Z`.

- `basic` — используется базовый парсер.

ClickHouse может парсить только базовый формат YYYY-MM-DD HH:MM:SS или YYYY-MM-DD. Например, 2019-08-20 10:18:56 или 2019-08-20.

Значение по умолчанию: basic.

См. также:

- [Тип данных DateTime](#).
- [Функции для работы с датой и временем](#).

## date\_time\_output\_format

Позволяет выбрать разные выходные форматы текстового представления даты и времени.

Возможные значения:

- simple - простой выходной формат.

Выходные дата и время Clickhouse в формате YYYY-MM-DD hh:mm:ss. Например, 2019-08-20 10:18:56. Расчет выполняется в соответствии с часовым поясом типа данных (если он есть) или часовым поясом сервера.

- iso - выходной формат ISO.

Выходные дата и время Clickhouse в формате ISO 8601 YYYY-MM-DDThh:mm:ssZ. Например, 2019-08-20T10:18:56Z. Обратите внимание, что выходные данные отображаются в формате UTC (Z означает UTC).

- unix\_timestamp - выходной формат Unix.

Выходные дата и время в формате Unix. Например 1566285536.

Значение по умолчанию: simple.

См. также:

- [Тип данных DateTime](#)
- [Функции для работы с датой и временем](#)

## join\_default\_strictness

Устанавливает строгость по умолчанию для JOIN.

Возможные значения:

- ALL — если в правой таблице несколько совпадающих строк, данные умножаются на количество этих строк. Это нормальное поведение JOIN как в стандартном SQL.
- ANY — если в правой таблице несколько соответствующих строк, то соединяется только первая найденная. Если в «правой» таблице есть не более одной подходящей строки, то результаты ANY и ALL совпадают.
- Пустая строка — если ALL или ANY не указаны в запросе, то ClickHouse генерирует исключение.

Значение по умолчанию: ALL.

## join\_algorithm

Определяет алгоритм выполнения запроса JOIN.

Возможные значения:

- `hash` — используется алгоритм соединения хешированием.
- `partial_merge` — используется алгоритм соединения слиянием сортированных списков.
- `prefer_partial_merge` — используется алгоритм соединения слиянием сортированных списков, когда это возможно.
- `auto` — сервер ClickHouse пытается на лету заменить алгоритм `hash` на `merge`, чтобы избежать переполнения памяти.

Значение по умолчанию: `hash`.

При использовании алгоритма `hash` правая часть `JOIN` загружается в оперативную память.

При использовании алгоритма `partial_merge` сервер сортирует данные и сбрасывает их на диск. Работа алгоритма `merge` в ClickHouse немного отличается от классической реализации. Сначала ClickHouse сортирует правую таблицу по блокам на основе **ключей соединения** и для отсортированных блоков строит индексы min-max. Затем он сортирует куски левой таблицы на основе ключей соединения и объединяет их с правой таблицей операцией `JOIN`. Созданные min-max индексы используются для пропуска тех блоков из правой таблицы, которые не участвуют в данной операции `JOIN`.

## join\_any\_take\_last\_row

Изменяет поведение операций, выполняемых со строгостью `ANY`.

### Внимание

Настройка применяется только для операций `JOIN`, выполняемых над таблицами с движком **Join**.

Возможные значения:

- 0 — если в правой таблице несколько соответствующих строк, то присоединяется только первая найденная строка.
- 1 — если в правой таблице несколько соответствующих строк, то присоединяется только последняя найденная строка.

Значение по умолчанию: 0.

См. также:

- [Секция JOIN](#)
- [Движок таблиц Join](#)
- [join\\_default\\_strictness](#)

## join\_use\_nulls

Устанавливает тип поведения `JOIN`. При объединении таблиц могут появиться пустые ячейки. ClickHouse заполняет их по-разному в зависимости от настроек.

Возможные значения:

- 0 — пустые ячейки заполняются значением по умолчанию соответствующего типа поля.

- 1 — `JOIN` ведёт себя как в стандартном SQL. Тип соответствующего поля преобразуется в `Nullable`, а пустые ячейки заполняются значениями `NULL`.

## partial\_merge\_join\_optimizations

Отключает все оптимизации для запросов `JOIN` с частичным MergeJoin алгоритмом.

По умолчанию оптимизации включены, что может привести к неправильным результатам. Если вы видите подозрительные результаты в своих запросах, отключите оптимизацию с помощью этого параметра. В различных версиях сервера ClickHouse, оптимизация может отличаться.

Возможные значения:

- 0 — Оптимизация отключена.
- 1 — Оптимизация включена.

Значение по умолчанию: 1.

## partial\_merge\_join\_rows\_in\_right\_blocks

Устанавливает предельные размеры блоков данных «правого» соединения, для запросов `JOIN` с частичным MergeJoin алгоритмом.

Сервер ClickHouse:

1. Разделяет данные правого соединения на блоки с заданным числом строк.
2. Индексирует для каждого блока минимальное и максимальное значение.
3. Выгружает подготовленные блоки на диск, если это возможно.

Возможные значения:

- Положительное целое число. Рекомендуемый диапазон значений [1000, 100000].

Значение по умолчанию: 65536.

## join\_on\_disk\_max\_files\_to\_merge

Устанавливает количество файлов, разрешенных для параллельной сортировки, при выполнении операций MergeJoin на диске.

Чем больше значение параметра, тем больше оперативной памяти используется и тем меньше используется диск (I/O).

Возможные значения:

- Положительное целое число, больше 2.

Значение по умолчанию: 64.

## temporary\_files\_codec

Устанавливает метод сжатия для временных файлов на диске, используемых при сортировки и объединения.

Возможные значения:

- LZ4 — применять сжатие, используя алгоритм `LZ4`
- NONE — не применять сжатие.

Значение по умолчанию: `LZ4`.

## any\_join\_distinct\_right\_table\_keys

Включает устаревшее поведение сервера ClickHouse при выполнении операций ANY INNER|LEFT JOIN.

### Внимание

Используйте этот параметр только в целях обратной совместимости, если ваши варианты использования требуют устаревшего поведения JOIN.

Когда включено устаревшее поведение:

- Результаты операций "t1 ANY LEFT JOIN t2" и "t2 ANY RIGHT JOIN t1" не равны, поскольку ClickHouse использует логику с сопоставлением ключей таблицы "многие к одному слева направо".
- Результаты операций ANY INNER JOIN содержат все строки из левой таблицы, аналогично операции SEMI LEFT JOIN.

Когда устаревшее поведение отключено:

- Результаты операций t1 ANY LEFT JOIN t2 и t2 ANY RIGHT JOIN t1 равно, потому что ClickHouse использует логику сопоставления ключей один-ко-многим в операциях ANY RIGHT JOIN.
- Результаты операций ANY INNER JOIN содержат по одной строке на ключ из левой и правой таблиц.

Возможные значения:

- 0 — Устаревшее поведение отключено.
- 1 — Устаревшее поведение включено.

Значение по умолчанию: 0.

См. также:

- [JOIN strictness](#)

## max\_block\_size

Данные в ClickHouse обрабатываются по блокам (наборам кусочков столбцов). Внутренние циклы обработки для одного блока достаточно эффективны, но есть заметные издержки на каждый блок. Настройка max\_block\_size — это рекомендация, какой размер блока (в количестве строк) загружать из таблиц. Размер блока не должен быть слишком маленьким, чтобы затраты на каждый блок были заметны, но не слишком велики, чтобы запрос с LIMIT, который завершается после первого блока, обрабатывался быстро. Цель состоит в том, чтобы не использовалось слишком много оперативки при вынимании большого количества столбцов в несколько потоков; чтобы оставалась хоть какая-нибудь кэш-локальность.

Значение по умолчанию: 65,536.

Из таблицы не всегда загружаются блоки размера max\_block\_size. Если ясно, что нужно прочитать меньше данных, то будет считан блок меньшего размера.

## preferred\_block\_size\_bytes

Служит для тех же целей что и max\_block\_size, но задает рекомендуемый размер блоков в байтах, выбирая адаптивное количество строк в блоке.

При этом размер блока не может быть более max\_block\_size строк.

По умолчанию: 1,000,000. Работает только при чтении из MergeTree-движков.

## `merge_tree_uniform_read_distribution`

При чтении из таблиц `MergeTree` ClickHouse использует несколько потоков. Этот параметр включает/выключает равномерное распределение заданий по рабочим потокам. Алгоритм равномерного распределения стремится сделать время выполнения всех потоков примерно равным для одного запроса `SELECT`.

Возможные значения:

- 0 — не использовать равномерное распределение заданий на чтение.
- 1 — использовать равномерное распределение заданий на чтение.

Значение по умолчанию: 1.

## `merge_tree_min_rows_for_concurrent_read`

Если количество строк, считываемых из файла таблицы `MergeTree` превышает `merge_tree_min_rows_for_concurrent_read`, то ClickHouse пытается выполнить одновременное чтение из этого файла в несколько потоков.

Возможные значения:

- Любое положительное целое число.

Значение по умолчанию: 163840.

## `merge_tree_min_bytes_for_concurrent_read`

Если число байтов, которое должно быть прочитано из одного файла таблицы с движком `MergeTree`, превышает значение `merge_tree_min_bytes_for_concurrent_read`, то ClickHouse выполняет одновременное чтение в несколько потоков из этого файла.

Возможное значение:

- Положительное целое число.

Значение по умолчанию: 251658240.

## `merge_tree_min_rows_for_seek`

Если расстояние между двумя блоками данных для чтения в одном файле меньше, чем `merge_tree_min_rows_for_seek` строк, то ClickHouse не перескакивает (`seek`) через блоки, а считывает данные последовательно.

Возможные значения:

- Положительное целое число.

Значение по умолчанию: 0.

## `merge_tree_min_bytes_for_seek`

Если расстояние между двумя блоками данных для чтения в одном файле меньше, чем `merge_tree_min_bytes_for_seek` байтов, то ClickHouse не перескакивает (`seek`) через блоки, а считывает данные последовательно.

Возможные значения:

- Положительное целое число.

Значение по умолчанию: 0.

## merge\_tree\_coarse\_index\_granularity

При поиске данных ClickHouse проверяет засечки данных в файле индекса. Если ClickHouse обнаруживает, что требуемые ключи находятся в некотором диапазоне, он делит этот диапазон на `merge_tree_coarse_index_granularity` поддиапазонов и выполняет в них рекурсивный поиск нужных ключей.

Возможные значения:

- Положительное целое число.

Значение по умолчанию: 8.

## merge\_tree\_max\_rows\_to\_use\_cache

Если требуется прочитать более, чем `merge_tree_max_rows_to_use_cache` строк в одном запросе, ClickHouse не используют кэш несжатых блоков.

Кэш несжатых блоков хранит данные, извлечённые при выполнении запросов. ClickHouse использует этот кэш для ускорения ответов на повторяющиеся небольшие запросы. Настройка защищает кэш от замусоривания запросами, для выполнения которых необходимо извлечь большое количество данных. Настройка сервера `uncompressed_cache_size` определяет размер кэша несжатых блоков.

Возможные значения:

- Положительное целое число.

Значение по умолчанию: 128 × 8192.

## merge\_tree\_max\_bytes\_to\_use\_cache

Если требуется прочитать более, чем `merge_tree_max_bytes_to_use_cache` байтов в одном запросе, ClickHouse не используют кэш несжатых блоков.

Кэш несжатых блоков хранит данные, извлечённые при выполнении запросов. ClickHouse использует кэш для ускорения ответов на повторяющиеся небольшие запросы. Настройка защищает кэш от переполнения. Настройка сервера `uncompressed_cache_size` определяет размер кэша несжатых блоков.

Возможные значения:

- Положительное целое число.

Значение по умолчанию: 2013265920.

## merge\_tree\_clear\_old\_temporary\_directories\_interval\_seconds

Задает интервал в секундах для удаления старых временных каталогов на сервере ClickHouse.

Возможные значения:

- Положительное целое число.

Значение по умолчанию: 60 секунд.

## merge\_tree\_clear\_old\_parts\_interval\_seconds

Задает интервал в секундах для удаления старых кусков данных, журналов предзаписи (WAL) и мутаций на сервере ClickHouse .

Возможные значения:

- Положительное целое число.

Значение по умолчанию: 1 секунда.

## min\_bytes\_to\_use\_direct\_io

Минимальный объём данных, необходимый для прямого (небуферизованного) чтения/записи (direct I/O) на диск.

ClickHouse использует этот параметр при чтении данных из таблиц. Если общий объём хранения всех данных для чтения превышает `min_bytes_to_use_direct_io` байт, тогда ClickHouse использует флаг `O_DIRECT` при чтении данных с диска.

Возможные значения:

- 0 — прямой ввод-вывод отключен.
- Положительное целое число.

Значение по умолчанию: 0.

## network\_compression\_method

Устанавливает метод сжатия данных, который используется для обмена данными между серверами и между сервером и `clickhouse-client`.

Возможные значения:

- LZ4 — устанавливает метод сжатия LZ4.
- ZSTD — устанавливает метод сжатия ZSTD.

Значение по умолчанию: LZ4.

### **См. также**

- [network\\_zstd\\_compression\\_level](#)

## network\_zstd\_compression\_level

Регулирует уровень сжатия ZSTD. Используется только тогда, когда `network_compression_method` установлен на ZSTD.

Возможные значения:

- Положительное целое число от 1 до 15.

Значение по умолчанию: 1.

## log\_queries

Установка логирования запроса.

Запросы, переданные в ClickHouse с этой настройкой, логируются согласно правилам конфигурационного параметра сервера `query_log`.

Пример:

```
log_queries=1
```

## log\_queries\_min\_query\_duration\_ms

Минимальное время выполнения запроса для логирования в системные таблицы:

- system.query\_log
- system.query\_thread\_log

В случае ненулевого порога `log_queries_min_query_duration_ms`, в лог будут записываться лишь события об окончании выполнения запроса:

- QUERY\_FINISH
- EXCEPTION\_WHILE\_PROCESSING
- Тип: milliseconds
- Значение по умолчанию: 0 (логировать все запросы)

## log\_queries\_min\_type

Задаёт минимальный уровень логирования в `query_log`.

Возможные значения:

- QUERY\_START (=1)
- QUERY\_FINISH (=2)
- EXCEPTION\_BEFORE\_START (=3)
- EXCEPTION\_WHILE\_PROCESSING (=4)

Значение по умолчанию: `QUERY_START`.

Можно использовать для ограничения того, какие объекты будут записаны в `query_log`, например, если вас интересуют ошибки, тогда вы можете использовать `EXCEPTION_WHILE_PROCESSING`:

```
log_queries_min_type='EXCEPTION_WHILE_PROCESSING'
```

## log\_query\_threads

Установка логирования информации о потоках выполнения запроса.

Лог информации о потоках выполнения запросов, переданных в ClickHouse с этой установкой, записывается согласно правилам конфигурационного параметра сервера `query_thread_log`.

Пример:

```
log_query_threads=1
```

## log\_comment

Задаёт значение поля `log_comment` таблицы `system.query_log` и текст комментария в логе сервера.

Может быть использована для улучшения читабельности логов сервера. Кроме того, помогает быстро выделить связанные с тестом запросы из `system.query_log` после запуска `clickhouse-test`.

Возможные значения:

- Любая строка не длиннее `max_query_size`. При превышении длины сервер сгенерирует исключение.

Значение по умолчанию: пустая строка.

## Пример

Запрос:

```
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;
```

Результат:

type	query
QueryStart	SELECT 1;
QueryFinish	SELECT 1;

## max\_insert\_block\_size

Формировать блоки указанного размера, при вставке в таблицу.

Эта настройка действует только в тех случаях, когда сервер сам формирует такие блоки.

Например, при INSERT-е через HTTP интерфейс, сервер парсит формат данных, и формирует блоки указанного размера.

А при использовании clickhouse-client, клиент сам парсит данные, и настройка `max_insert_block_size` на сервере не влияет на размер вставляемых блоков.

При использовании INSERT SELECT, настройка так же не имеет смысла, так как данные будут вставляться теми блоками, которые вышли после SELECT-а.

Значение по умолчанию: 1,048,576.

Это значение намного больше, чем `max_block_size`. Это сделано, потому что некоторые движки таблиц (`*MergeTree`) будут на каждый вставляемый блок формировать кусок данных на диске, что является довольно большой сущностью. Также, в таблицах типа `*MergeTree`, данные сортируются при вставке, и достаточно большой размер блока позволяет отсортировать больше данных в оперативке.

## min\_insert\_block\_size\_rows

Устанавливает минимальное количество строк в блоке, который может быть вставлен в таблицу запросом `INSERT`. Блоки меньшего размера склеиваются в блоки большего размера.

Возможные значения:

- Целое положительное число.
- 0 — Склейка блоков выключена.

Значение по умолчанию: 1048576.

## min\_insert\_block\_size\_bytes

Устанавливает минимальное количество байтов в блоке, который может быть вставлен в таблицу запросом `INSERT`. Блоки меньшего размера склеиваются в блоки большего размера.

Возможные значения:

- Целое положительное число.
- 0 — Склейка блоков выключена.

Значение по умолчанию: 268435456.

## max\_replica\_delay\_for\_distributed\_queries

Отключает отстающие реплики при распределенных запросах. См. [Репликация](#).

Устанавливает время в секундах. Если отставание реплики больше установленного значения, то реплика не используется.

Значение по умолчанию: 300.

Используется при выполнении `SELECT` из распределенной таблицы, которая указывает на реплицированные таблицы.

## max\_threads

Максимальное количество потоков обработки запроса без учёта потоков для чтения данных с удалённых серверов (смотрите параметр `max_distributed_connections`).

Этот параметр относится к потокам, которые выполняют параллельно одни стадии конвейера выполнения запроса.

Например, при чтении из таблицы, если есть возможность вычислять выражения с функциями, фильтровать с помощью `WHERE` и предварительно агрегировать для `GROUP BY` параллельно, используя хотя бы количество потоков `max_threads`, то используются `max_threads`.

Значение по умолчанию: количество процессорных ядер без учёта Hyper-Threading.

Для запросов, которые быстро завершаются из-за `LIMIT`-а, имеет смысл выставить `max_threads` поменьше. Например, если нужное количество записей находится в каждом блоке, то при `max_threads = 8` будет считано 8 блоков, хотя достаточно было прочитать один.

Чем меньше `max_threads`, тем меньше будет использоваться оперативки.

## max\_insert\_threads

Максимальное количество потоков для выполнения запроса `INSERT SELECT`.

Возможные значения:

- 0 (или 1) — `INSERT SELECT` не выполняется параллельно.
- Положительное целое число, больше 1.

Значение по умолчанию: 0.

Параллельный `INSERT SELECT` действует только в том случае, если часть `SELECT` выполняется параллельно, см. настройку `max_threads`.

Чем больше значение `max_insert_threads`, тем больше потребление оперативной памяти.

## max\_compress\_block\_size

Максимальный размер блоков несжатых данных перед сжатием при записи в таблицу. По умолчанию - 1 048 576 (1 MiB). При уменьшении размера, незначительно уменьшается коэффициент сжатия, незначительно возрастает скорость сжатия и разжатия за счёт кэш-локальности, и уменьшается потребление оперативной памяти.

## Предупреждение

Эта настройка экспертного уровня, не используйте ее, если вы только начинаете работать с Clickhouse.

Не путайте блоки для сжатия (кусок памяти, состоящий из байт) и блоки для обработки запроса (пачка строк из таблицы).

### min\_compress\_block\_size

Для таблиц типа **MergeTree**. В целях уменьшения задержек при обработке запросов, блок сжимается при записи следующей засечки, если его размер не меньше `min_compress_block_size`. По умолчанию - 65 536.

Реальный размер блока, если несжатых данных меньше `max_compress_block_size`, будет не меньше этого значения и не меньше объёма данных на одну засечку.

Рассмотрим пример. Пусть `index_granularity`, указанная при создании таблицы - 8192.

Пусть мы записываем столбец типа `UInt32` (4 байта на значение). При записи 8192 строк, будет всего 32 КБ данных. Так как `min_compress_block_size = 65 536`, сжатый блок будет сформирован на каждые две засечки.

Пусть мы записываем столбец `URL` типа `String` (средний размер - 60 байт на значение). При записи 8192 строк, будет, в среднем, чуть меньше 500 КБ данных. Так как это больше 65 536 строк, то сжатый блок будет сформирован на каждую засечку. В этом случае, при чтении с диска данных из диапазона в одну засечку, не будет разжато лишних данных.

## Предупреждение

Эта настройка экспертного уровня, не используйте ее, если вы только начинаете работать с Clickhouse.

### max\_query\_size

Максимальный кусок запроса, который будет считан в оперативку для разбора парсером языка SQL. Запрос `INSERT` также содержит данные для `INSERT-a`, которые обрабатываются отдельным, потоковым парсером (расходящим  $O(1)$  оперативки), и не учитываются в этом ограничении.

Значение по умолчанию: 256 Кб.

### max\_parser\_depth

Ограничивает максимальную глубину рекурсии в парсере рекурсивного спуска. Позволяет контролировать размер стека.

Возможные значения:

- Положительное целое число.
- 0 — Глубина рекурсии не ограничена.

Значение по умолчанию: 1000.

### interactive\_delay

Интервал в микросекундах для проверки, не запрошена ли остановка выполнения запроса, и отправки прогресса.

Значение по умолчанию: 100,000 (проверять остановку запроса и отправлять прогресс десять раз в секунду).

## connect\_timeout, receive\_timeout, send\_timeout

Таймауты в секундах на сокет, по которому идёт общение с клиентом.

Значение по умолчанию: 10, 300, 300.

## cancel\_http\_READONLY\_queries\_on\_client\_close

Отменяет HTTP readonly запросы (например, SELECT), когда клиент обрывается соединение до завершения получения данных.

Значение по умолчанию: 0

## poll\_interval

Блокироваться в цикле ожидания запроса в сервере на указанное количество секунд.

Значение по умолчанию: 10.

## max\_distributed\_connections

Максимальное количество одновременных соединений с удалёнными серверами при распределённой обработке одного запроса к одной таблице типа Distributed. Рекомендуется выставлять не меньше, чем количество серверов в кластере.

Значение по умолчанию: 1024.

Следующие параметры имеют значение только на момент создания таблицы типа Distributed (и при запуске сервера), поэтому их не имеет смысла менять в рантайме.

## distributed\_connections\_pool\_size

Максимальное количество одновременных соединений с удалёнными серверами при распределённой обработке всех запросов к одной таблице типа Distributed. Рекомендуется выставлять не меньше, чем количество серверов в кластере.

Значение по умолчанию: 1024.

## max\_distributed\_depth

Ограничивает максимальную глубину рекурсивных запросов для **Distributed** таблиц.

Если значение превышено, сервер генерирует исключение.

Возможные значения:

- Положительное целое число.
- 0 — глубина не ограничена.

Значение по умолчанию: 5.

## max\_replicated\_fetches\_network\_bandwidth\_for\_server

Ограничивает максимальную скорость обмена данными в сети (в байтах в секунду) для синхронизации между **репликами**. Применяется только при запуске сервера. Можно также ограничить скорость для конкретной таблицы с помощью настройки **max\_replicated\_fetches\_network\_bandwidth**.

Значение настройки соблюдается неточно.

Возможные значения:

- Любое целое положительное число.
- 0 — Скорость не ограничена.

Значение по умолчанию: 0.

### **Использование**

Может быть использована для ограничения скорости сети при репликации данных для добавления или замены новых узлов.

## **max\_replicated\_sends\_network\_bandwidth\_for\_server**

Ограничивает максимальную скорость обмена данными в сети (в байтах в секунду) для **репликационных** отправок. Применяется только при запуске сервера. Можно также ограничить скорость для конкретной таблицы с помощью настройки **max\_replicated\_sends\_network\_bandwidth**.

Значение настройки соблюдается неточно.

Возможные значения:

- Любое целое положительное число.
- 0 — Скорость не ограничена.

Значение по умолчанию: 0.

### **Использование**

Может быть использована для ограничения скорости сети при репликации данных для добавления или замены новых узлов.

## **connect\_timeout\_with\_failover\_ms**

Таймаут в миллисекундах на соединение с удалённым сервером, для движка таблиц Distributed, если используются секции shard и replica в описании кластера.

В случае неуспеха, делается несколько попыток соединений с разными репликами.

Значение по умолчанию: 50.

## **connection\_pool\_max\_wait\_ms**

Время ожидания соединения в миллисекундах, когда пул соединений заполнен.

Возможные значения:

- Положительное целое число.
- 0 — Бесконечный таймаут.

Значение по умолчанию: 0.

## `connections_with_failover_max_tries`

Максимальное количество попыток соединения с каждой репликой, для движка таблиц Distributed.

Значение по умолчанию: 3.

## `extremes`

Считать ли экстремальные значения (минимумы и максимумы по столбцам результата запроса).

Принимает 0 или 1. По умолчанию - 0 (выключено).

Подробнее смотрите раздел «[Экстремальные значения](#)».

## `kafka_max_wait_ms`

Время ожидания в миллисекундах для чтения сообщений из [Kafka](#) перед повторной попыткой.

Возможные значения:

- Положительное целое число.
- 0 — Бесконечный таймаут.

Значение по умолчанию: 5000.

См. также:

- [Apache Kafka](#)

## `use_uncompressed_cache`

Использовать ли кэш разжатых блоков. Принимает 0 или 1. По умолчанию - 0 (выключено).

Использование кэша несжатых блоков (только для таблиц семейства MergeTree) может существенно сократить задержку и увеличить пропускную способность при работе с большим количеством коротких запросов. Включите эту настройку для пользователей, от которых идут частые короткие запросы. Также обратите внимание на конфигурационный параметр [uncompressed\\_cache\\_size](#) (настраивается только в конфигурационном файле) – размер кэша разжатых блоков. По умолчанию - 8 GiB. Кэш разжатых блоков заполняется по мере необходимости, а наиболее невостребованные данные автоматически удаляются.

Для запросов, читающих хоть немного приличный объём данных (миллион строк и больше), кэш разжатых блоков автоматически выключается, чтобы оставить место для действительно мелких запросов. Поэтому, можно держать настройку `use_uncompressed_cache` всегда выставленной в 1.

## `replace_running_query`

При использовании интерфейса HTTP может быть передан параметр `query_id`. Это любая строка, которая служит идентификатором запроса.

Если в этот момент, уже существует запрос от того же пользователя с тем же `query_id`, то поведение определяется параметром `replace_running_query`.

0 - (по умолчанию) кинуть исключение (не давать выполнить запрос, если запрос с таким же `query_id` уже выполняется);

1 - отменить старый запрос и начать выполнять новый.

Эта настройка, выставленная в 1, используется в Яндекс.Метрике для реализации suggest-а значений для условий сегментации. После ввода очередного символа, если старый запрос ещё не выполнился, его следует отменить.

## `replace_running_query_max_wait_ms`

Время ожидания завершения выполнения запроса с тем же `query_id`, когда активирована настройка `replace_running_query`.

Возможные значения:

- Положительное целое число.
- 0 — Создание исключения, которое не позволяет выполнить новый запрос, если сервер уже выполняет запрос с тем же `query_id`.

Значение по умолчанию: 5000.

## `stream_flush_interval_ms`

Работает для таблиц со стриммингом в случае тайм-аута, или когда поток генерирует `max_insert_block_size` строк.

Значение по умолчанию: 7500.

Чем меньше значение, тем чаще данные сбрасываются в таблицу. Установка слишком низкого значения приводит к снижению производительности.

## `load_balancing`

Задает алгоритм выбора реплик, используемый при обработке распределенных запросов.

ClickHouse поддерживает следующие алгоритмы выбора реплик:

- `Random` (by default)
- `Nearest hostname`
- `In order`
- `First or random`
- `Round robin`

См. также:

- [`distributed\_replica\_max\_ignored\_errors`](#)

### `Random (by Default)`

```
load_balancing = random
```

Для каждой реплики считается количество ошибок. Запрос отправляется на реплику с минимальным числом ошибок, а если таких несколько, то на случайную из них.

Недостатки: не учитывается близость серверов; если на репликах оказались разные данные, то вы будете получать так же разные данные.

### `Nearest Hostname`

```
load_balancing = nearest_hostname
```

Для каждой реплики считается количество ошибок. Каждые 5 минут, число ошибок целочисленно делится на 2. Таким образом, обеспечивается расчёт числа ошибок за недавнее время с экспоненциальным сглаживанием. Если есть одна реплика с минимальным числом ошибок (то есть, на других репликах недавно были ошибки) - запрос отправляется на неё. Если есть несколько реплик с одинаковым минимальным числом ошибок, то запрос отправляется на реплику, имя хоста которой в конфигурационном файле минимально отличается от имени хоста сервера (по количеству отличающихся символов на одинаковых позициях, до минимальной длины обеих имён хостов).

Для примера, `example01-01-1` и `example01-01-2.yandex.ru` отличаются в одной позиции, а `example01-01-1` и `example01-02-2` - в двух.

Этот метод может показаться примитивным, но он не требует внешних данных о топологии сети и не сравнивает IP-адреса, что было бы сложно для наших IPv6-адресов.

Таким образом, если есть равнозначные реплики, предпочтается ближайшая по имени.

Также можно сделать предположение, что при отправке запроса на один и тот же сервер, в случае отсутствия сбоев, распределённый запрос будет идти тоже на одни и те же серверы. То есть, даже если на репликах расположены разные данные, запрос будет возвращать в основном одинаковые результаты.

## In Order

```
load_balancing = in_order
```

Реплики с одинаковым количеством ошибок опрашиваются в порядке, определённом конфигурацией.

Этот способ подходит для тех случаев, когда вы точно знаете, какая реплика предпочтительнее.

## First or Random

```
load_balancing = first_or_random
```

Алгоритм выбирает первую реплику или случайную реплику, если первая недоступна. Он эффективен в топологиях с перекрестной репликацией, но бесполезен в других конфигурациях.

Алгоритм `first or random` решает проблему алгоритма `in order`. При использовании `in order`, если одна реплика перестаёт отвечать, то следующая за ней принимает двойную нагрузку, в то время как все остальные обрабатывают свой обычный трафик. Алгоритм `first or random` равномерно распределяет нагрузку между репликами.

## Round Robin

```
load_balancing = round_robin
```

Этот алгоритм использует циклический перебор реплик с одинаковым количеством ошибок (учитываются только запросы с алгоритмом `round_robin`).

## `prefer_localhost_replica`

Включает или выключает предпочтительное использование `localhost` реплики при обработке распределенных запросов.

Возможные значения:

- 1 — ClickHouse всегда отправляет запрос на `localhost` реплику, если она существует.

- 0 — ClickHouse использует балансировку, заданную настройкой [load\\_balancing](#).

Значение по умолчанию: 1.

## Warning

Отключайте эту настройку при использовании [max\\_parallel\\_replicas](#).

## totals\_mode

Каким образом вычислять TOTALS при наличии HAVING, а также при наличии max\_rows\_to\_group\_by и group\_by\_overflow\_mode = 'any'.

Смотрите раздел «Модификатор WITH TOTALS».

## totals\_auto\_threshold

Порог для totals\_mode = 'auto'.

Смотрите раздел «Модификатор WITH TOTALS».

## max\_parallel\_replicas

Максимальное количество используемых реплик каждого шарда при выполнении запроса.

Возможные значения:

- Целое положительное число.

### Дополнительная информация

Эта настройка полезна для реплицируемых таблиц с ключом сэмплирования. Запрос может обрабатываться быстрее, если он выполняется на нескольких серверах параллельно. Однако производительность обработки запроса, наоборот, может упасть в следующих ситуациях:

- Позиция ключа сэмплирования в ключеpartitionирования не позволяет выполнять эффективное сканирование.
- Добавление ключа сэмплирования в таблицу делает фильтрацию по другим столбцам менее эффективной.
- Ключ сэмплирования является выражением, которое сложно вычисляется.
- У распределения сетевых задержек в кластере длинный «хвост», из-за чего при параллельных запросах к нескольким серверам увеличивается среднее время задержки.

## Предупреждение

Параллельное выполнение запроса может привести к неверному результату, если в запросе есть объединение или подзапросы и при этом таблицы не удовлетворяют определенным требованиям. Подробности смотрите в разделе [Распределенные подзапросы и max\\_parallel\\_replicas](#).

## compile\_expressions

Включает или выключает компиляцию часто используемых функций и операторов. Компиляция производится в нативный код платформы с помощью LLVM во время выполнения.

Возможные значения:

- 0 — компиляция выключена.
- 1 — компиляция включена.

Значение по умолчанию: 1.

## min\_count\_to\_compile\_expression

Минимальное количество выполнений одного и того же выражения до его компиляции.

Значение по умолчанию: 3.

## input\_format\_skip\_unknown\_fields

Если значение равно true, то при выполнении INSERT входные данные из столбцов с неизвестными именами будут пропущены. В противном случае эта ситуация создаст исключение.

Работает для форматов JSONEachRow и TSKV.

## output\_format\_json\_quote\_64bit\_integers

Управляет кавычками при выводе 64-битных или более целых чисел (например, UInt64 или Int128) в формате JSON.

По умолчанию такие числа заключаются в кавычки. Это поведение соответствует большинству реализаций JavaScript.

Возможные значения:

- 0 — числа выводятся без кавычек.
- 1 — числа выводятся в кавычках.

Значение по умолчанию: 1.

## output\_format\_json\_quote\_denormals

При выводе данных в формате JSON включает отображение значений +nan, -nan, +inf, -inf.

Возможные значения:

- 0 — выключена.
- 1 — включена.

Значение по умолчанию: 0.

### Пример

Рассмотрим следующую таблицу account\_orders:

id	name	duration	period	area
1	Andrew	20	0	400
2	John	40	0	0
3	Bob	15	0	-100

Когда output\_format\_json\_quote\_denormals = 0, следующий запрос возвращает значения null.

```
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
    }
}
```

Если `output_format_json_quote_denormals = 1`, то запрос вернет:

```
{
    "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

Символ, интерпретируемый как разделитель в данных формата CSV. По умолчанию — `,`.

## input\_format\_csv\_unquoted\_null\_literal\_as\_null

Для формата CSV включает или выключает парсинг неэкранированной строки `NULL` как литерала (синоним для `\N`)

## **input\_format\_csv\_enum\_as\_number**

Включает или отключает парсинг значений перечислений как идентификаторов перечислений для входного формата CSV.

Возможные значения:

- 0 — парсинг значений перечисления как значений.
- 1 — парсинг значений перечисления как идентификаторов перечисления.

Значение по умолчанию: 0.

### **Пример**

Рассмотрим таблицу:

```
CREATE TABLE table_with_enum_column_for_csv_insert (Id Int32,Value Enum('first' = 1, 'second' = 2))
ENGINE=Memory();
```

При включенной настройке `input_format_csv_enum_as_number`:

```
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;
```

Результат:

Id	Value
102	second

При отключенной настройке `input_format_csv_enum_as_number` запрос `INSERT`:

```
SET input_format_csv_enum_as_number = 0;
INSERT INTO table_with_enum_column_for_csv_insert FORMAT CSV 102,2;
```

сгенерирует исключение.

## **output\_format\_csv\_crlf\_end\_of\_line**

Использовать в качестве разделителя строк для CSV формата CRLF (DOS/Windows стиль) вместо LF (Unix стиль).

## **output\_format\_tsv\_crlf\_end\_of\_line**

Использовать в качестве разделителя строк для TSV формата CRLF (DOC/Windows стиль) вместо LF (Unix стиль).

## **insert\_quorum**

Включает кворумную запись.

- Если `insert_quorum < 2`, то кворумная запись выключена.
- Если `insert_quorum >= 2`, то кворумная запись включена.

Значение по умолчанию: 0.

## Кворумная запись

`INSERT` завершается успешно только в том случае, когда ClickHouse смог без ошибки записать данные в `insert_quorum` реплик за время `insert_quorum_timeout`. Если по любой причине количество реплик с успешной записью не достигнет `insert_quorum`, то запись считается не состоявшейся и ClickHouse удалит вставленный блок из всех реплик, куда уже успел записать данные.

Все реплики в кворуме консистентны, т.е. содержат данные всех более ранних запросов `INSERT`. Последовательность `INSERT` линеаризуется.

При чтении данных, записанных с `insert_quorum` можно использовать настройку `select_sequential_consistency`.

ClickHouse генерирует исключение

- Если количество доступных реплик на момент запроса меньше `insert_quorum`.
- При попытке записать данные в момент, когда предыдущий блок ещё не вставлен в `insert_quorum` реплик. Эта ситуация может возникнуть, если пользователь вызвал `INSERT` прежде, чем завершился предыдущий с `insert_quorum`.

См. также:

- [insert\\_quorum\\_timeout](#)
- [select\\_sequential\\_consistency](#)

## insert\_quorum\_timeout

Время ожидания кворумной записи в миллисекундах. Если время прошло, а запись так не состоялась, то ClickHouse сгенерирует исключение и клиент должен повторить запрос на запись того же блока на эту же или любую другую реплику.

Значение по умолчанию: 600 000 миллисекунд (10 минут).

См. также:

- [insert\\_quorum](#)
- [select\\_sequential\\_consistency](#)

## select\_sequential\_consistency

Включает или выключает последовательную консистентность для запросов `SELECT`.

Возможные значения:

- 0 — выключена.
- 1 — включена.

Значение по умолчанию: 0.

### Использование

Когда последовательная консистентность включена, то ClickHouse позволит клиенту выполнить запрос `SELECT` только к тем репликам, которые содержат данные всех предыдущих запросов `INSERT`, выполненных с `insert_quorum`. Если клиент обратится к неполной реплике, то ClickHouse сгенерирует исключение. В запросе `SELECT` не будут участвовать данные, которые ещё не были записаны на кворум реплик.

См. также:

- [insert\\_quorum](#)
- [insert\\_quorum\\_timeout](#)

## insert\_deduplicate

Включает и выключает дедупликацию для запросов `INSERT` (для Replicated\* таблиц).

Возможные значения:

- 0 — выключена.
- 1 — включена.

Значение по умолчанию: 1.

По умолчанию блоки, вставляемые в реплицируемые таблицы оператором `INSERT`, дедуплицируются (см. [Репликация данных](#)).

## deduplicate\_blocks\_in\_dependent\_materialized\_views

Включает и выключает проверку дедупликации для материализованных представлений, которые получают данные из Replicated\* таблиц.

Возможные значения:

- 0 — выключена.
- 1 — включена.

Значение по умолчанию: 0.

По умолчанию проверка дедупликации у материализованных представлений не производится, а наследуется от Replicated\* (основной) таблицы, за которой «следит» материализованное представление.

Т.е. если `INSERT` в основную таблицу д.б. пропущен (сдедуплицирован), то автоматически не будет вставки и в материализованные представления. Это имплементировано для того, чтобы работали материализованные представления, которые сильно группируют данные основных `INSERT`, до такой степени что блоки вставляемые в материализованные представления получаются одинаковыми для разных `INSERT` в основную таблицу.

Одновременно это «ломает» идемпотентность вставки в материализованные представления. Т.е. если `INSERT` был успешен в основную таблицу и неуспешен в таблицу материализованного представления (напр. из-за сетевого сбоя при коммуникации с Zookeeper), клиент получит ошибку и попытается повторить `INSERT`. Но вставки в материализованные представления произведено не будет, потому что дедупликация сработает на основной таблице. Настройка `deduplicate_blocks_in_dependent_materialized_views` позволяет это изменить. Т.е. при повторном `INSERT` будет произведена дедупликация на таблице материализованного представления, и повторный инсерт вставит данные в таблицу материализованного представления, которые не удалось вставить из-за сбоя первого `INSERT`.

## count\_distinct\_implementation

Задаёт, какая из функций `uniq*` используется при выполнении конструкции `COUNT(DISTINCT ...)`.

Возможные значения:

- `uniq`

- uniqCombined
- uniqCombined64
- uniqHLL12
- uniqExact

Значение по умолчанию: uniqExact.

## max\_network\_bytes

Ограничивает объём данных (в байтах), который принимается или передается по сети при выполнении запроса. Параметр применяется к каждомуциальному запросу.

Возможные значения:

- Положительное целое число.
- 0 — контроль объёма данных отключен.

Значение по умолчанию: 0.

## max\_network\_bandwidth

Ограничивает скорость обмена данными по сети в байтах в секунду. Параметр применяется к каждомуциальному запросу.

Возможные значения:

- Положительное целое число.
- 0 — контроль скорости передачи данных отключен.

Значение по умолчанию: 0.

## max\_network\_bandwidth\_for\_user

Ограничивает скорость обмена данными по сети в байтах в секунду. Этот параметр применяется ко всем одновременно выполняемым запросам, запущенным одним пользователем.

Возможные значения:

- Положительное целое число.
- 0 — управление скоростью передачи данных отключено.

Значение по умолчанию: 0.

## max\_network\_bandwidth\_for\_all\_users

Ограничивает скорость обмена данными по сети в байтах в секунду. Этот параметр применяется ко всем одновременно выполняемым запросам на сервере.

Возможные значения:

- Положительное целое число.
- 0 — управление скоростью передачи данных отключено.

Значение по умолчанию: 0.

## `skip_unavailable_shards`

Включает или отключает тихий пропуск недоступных шардов.

Шард считается недоступным, если все его реплики недоступны. Реплика недоступна в следующих случаях:

- ClickHouse не может установить соединение с репликой по любой причине.

ClickHouse предпринимает несколько попыток подключиться к реплике. Если все попытки оказались неудачными, реплика считается недоступной.

- Реплика не может быть разрешена с помощью DNS.

Если имя хоста реплики не может быть разрешено с помощью DNS, это может указывать на следующие ситуации:

- Нет записи DNS для хоста. Это может происходить в системах с динамическим DNS, например, [Kubernetes](<https://kubernetes.io>), где отключенные ноды не разрешаются с помощью DNS и это не ошибка.
- Ошибка конфигурации. Конфигурационный файл ClickHouse может содержать неправильное имя хоста.

Возможные значения:

- 1 — пропуск включен.

Если шард недоступен, то ClickHouse возвращает результат, основанный на неполных данных и не оповещает о проблемах с доступностью хостов.

- 0 — пропуск выключен.

Если шард недоступен, то ClickHouse генерирует исключение.

Значение по умолчанию: 0.

## `distributed_push_down_limit`

Включает или отключает **LIMIT**, применяемый к каждому шарду по отдельности.

Это позволяет избежать:

- отправки дополнительных строк по сети;
- обработки строк за пределами ограничения для инициатора.

Начиная с версии 21.9 вы больше не сможете получить неточные результаты, так как `distributed_push_down_limit` изменяет выполнение запроса только в том случае, если выполнено хотя бы одно из условий:

- `distributed_group_by_no_merge > 0`.
- запрос **не содержит** `GROUP BY/DISTINCT/LIMIT BY`, но содержит `ORDER BY/LIMIT`.
- запрос **содержит** `GROUP BY/DISTINCT/LIMIT BY` с `ORDER BY/LIMIT` и:
- включена настройка **optimize\_skip\_unused\_shards**.
- включена настройка `optimize_distributed_group_by_sharding_key`.

Возможные значения:

- 0 — выключена.
- 1 — включена.

Значение по умолчанию: 1.

См. также:

- [optimize\\_skip\\_unused\\_shards](#)

## optimize\_skip\_unused\_shards

Включает или отключает пропуск неиспользуемых шардов для запросов `SELECT`, в которых условие ключа шардирования задано в секции `WHERE/PREWHERE`. Предполагается, что данные распределены с помощью ключа шардирования, в противном случае запрос выдаст неверный результат.

Возможные значения:

- 0 — Выключена.
- 1 — Включена.

Значение по умолчанию: 0

## optimize\_skip\_unused\_shards\_nesting

Контролирует настройку `optimize_skip_unused_shards` (поэтому все еще требует `optimize_skip_unused_shards`) в зависимости от вложенности распределенного запроса (когда у вас есть `Distributed` таблица которая смотрит на другую `Distributed` таблицу).

Возможные значения:

- 0 — Выключена, `optimize_skip_unused_shards` работает всегда.
- 1 — Включает `optimize_skip_unused_shards` только для 1-ого уровня вложенности.
- 2 — Включает `optimize_skip_unused_shards` для 1-ого и 2-ого уровня вложенности.

Значение по умолчанию: 0

## force\_optimize\_skip\_unused\_shards

Разрешает или запрещает выполнение запроса, если настройка `optimize_skip_unused_shards` включена, а пропуск неиспользуемых шардов невозможен. Если данная настройка включена и пропуск невозможен, ClickHouse генерирует исключение.

Возможные значения:

- 0 — Выключена, `force_optimize_skip_unused_shards` работает всегда.
- 1 — Включает `force_optimize_skip_unused_shards` только для 1-ого уровня вложенности.
- 2 — Включает `force_optimize_skip_unused_shards` для 1-ого и 2-ого уровня вложенности.

Значение по умолчанию: 0

## force\_optimize\_skip\_unused\_shards\_nesting

Контролирует настройку `force_optimize_skip_unused_shards` (поэтому все еще требует `optimize_skip_unused_shards`) в зависимости от вложенности распределенного запроса (когда у вас есть `Distributed` таблица которая смотрит на другую `Distributed` таблицу).

Возможные значения:

- 0 - Выключена, `force_optimize_skip_unused_shards` работает всегда.

- 1 — Включает `force_optimize_skip_unused_shards` только для 1-ого уровня вложенности.
- 2 — Включает `force_optimize_skip_unused_shards` для 1-ого и 2-ого уровня вложенности.

Значение по умолчанию: 0

## `force_optimize_skip_unused_shards_no_nested`

Сбрасывает `optimize_skip_unused_shards` для вложенных `Distributed` таблиц.

Возможные значения:

- 1 — Включена.
- 0 — Выключена.

Значение по умолчанию: 0

## `optimize_throw_if_noop`

Включает или отключает генерирование исключения в случаях, когда запрос `OPTIMIZE` не выполняет мёрж.

По умолчанию, `OPTIMIZE` завершается успешно и в тех случаях, когда он ничего не сделал. Настройка позволяет отделить подобные случаи и включает генерирование исключения с поясняющим сообщением.

Возможные значения:

- 1 — генерирование исключения включено.
- 0 — генерирование исключения выключено.

Значение по умолчанию: 0.

## `optimize_functions_to_subcolumns`

Включает или отключает оптимизацию путем преобразования некоторых функций к чтению подстолбцов, таким образом уменьшая объем данных для чтения.

Могут быть преобразованы следующие функции:

- `length` к чтению подстолбца `size0` subcolumn.
- `empty` к чтению подстолбца `size0` subcolumn.
- `notEmpty` к чтению подстолбца `size0`.
- `isNull` к чтению подстолбца `null`.
- `isNotNull` к чтению подстолбца `null`.
- `count` к чтению подстолбца `null`.
- `mapKeys` к чтению подстолбца `keys`.
- `mapValues` к чтению подстолбца `values`.

Возможные значения:

- 0 — оптимизация отключена.
- 1 — оптимизация включена.

Значение по умолчанию: 0.

## optimize\_trivial\_count\_query

Включает или отключает оптимизацию простого запроса SELECT count() FROM table с использованием метаданных MergeTree. Если вы хотите управлять безопасностью на уровне строк, отключите оптимизацию.

Возможные значения:

- 0 — оптимизация отключена.
- 1 — оптимизация включена.

Значение по умолчанию: 1.

См. также:

- [optimize\\_functions\\_to\\_subcolumns](#)

## distributed\_replica\_error\_half\_life

- Тип: секунды
- Значение по умолчанию: 60 секунд

Управляет скоростью обнуления счетчика ошибок в распределенных таблицах. Предположим, реплика остается недоступна в течение какого-то времени, и за этот период накопилось 5 ошибок. Если настройка `distributed_replica_error_half_life` установлена в значение 1 секунда, то реплика снова будет считаться доступной через 3 секунды после последней ошибки.

См. также:

- [load\\_balancing](#)
- [Table engine Distributed](#)
- [distributed\\_replica\\_error\\_cap](#)
- [distributed\\_replica\\_max\\_ignored\\_errors](#)

## distributed\_replica\_error\_cap

- Тип: unsigned int
- Значение по умолчанию: 1000

Счетчик ошибок каждой реплики ограничен этим значением, чтобы одна реплика не накапливала слишком много ошибок.

См. также:

- [load\\_balancing](#)
- [Table engine Distributed](#)
- [distributed\\_replica\\_error\\_half\\_life](#)
- [distributed\\_replica\\_max\\_ignored\\_errors](#)

## distributed\_replica\_max\_ignored\_errors

- Тип: unsigned int
- Значение по умолчанию: 0

Количество ошибок, которые будут проигнорированы при выборе реплик (согласно алгоритму `load_balancing`).

См. также:

- [load\\_balancing](#)
- [Table engine Distributed](#)
- [distributed\\_replica\\_error\\_cap](#)
- [distributed\\_replica\\_error\\_half\\_life](#)

## `distributed_directory_monitor_sleep_time_ms`

Основной интервал отправки данных движком таблиц [Distributed](#). Фактический интервал растёт экспоненциально при возникновении ошибок.

Возможные значения:

- Положительное целое количество миллисекунд.

Значение по умолчанию: 100 миллисекунд.

## `distributed_directory_monitor_max_sleep_time_ms`

Максимальный интервал отправки данных движком таблиц [Distributed](#). Ограничивает экспоненциальный рост интервала, установленного настройкой [distributed\\_directory\\_monitor\\_sleep\\_time\\_ms](#).

Возможные значения:

- Положительное целое количество миллисекунд.

Значение по умолчанию: 30000 миллисекунд (30 секунд).

## `distributed_directory_monitor_batch_inserts`

Включает/выключает пакетную отправку вставленных данных.

Если пакетная отправка включена, то движок таблиц [Distributed](#) вместо того, чтобы отправлять каждый файл со вставленными данными по отдельности, старается отправить их все за одну операцию. Пакетная отправка улучшает производительность кластера за счет более оптимального использования ресурсов сервера и сети.

Возможные значения:

- 1 — включено.
- 0 — выключено.

Значение по умолчанию: 0.

## `os_thread_priority`

Устанавливает приоритет ([nice](#)) для потоков, исполняющих запросы. Планировщик ОС учитывает эти приоритеты при выборе следующего потока для исполнения на доступном ядре CPU.

## Предупреждение

Для использования этой настройки необходимо установить свойство `CAP_SYS_NICE`. Пакет `clickhouse-server` устанавливает его во время инсталляции. Некоторые виртуальные окружения не позволяют установить `CAP_SYS_NICE`. В этом случае, `clickhouse-server` выводит сообщение при запуске.

Допустимые значения:

- Любое значение из диапазона [-20, 19].

Более низкие значения означают более высокий приоритет. Потоки с низкими значениями приоритета `nice` выполняются чаще, чем потоки с более высокими значениями. Высокие значения предпочтительно использовать для долгих неинтерактивных запросов, поскольку это позволяет быстро выделить ресурс в пользу коротких интерактивных запросов.

Значение по умолчанию: 0.

## query\_profiler\_real\_time\_period\_ns

Устанавливает период для таймера реального времени [профилировщика запросов](#). Таймер реального времени считает wall-clock time.

Возможные значения:

- Положительное целое число в наносекундах.

Рекомендуемые значения:

- 10000000 (100 раз в секунду) наносекунд и меньшее значение для одиночных запросов.
- 1000000000 (раз в секунду) для профилирования в масштабе кластера.

- 0 для выключения таймера.

Тип: `UInt64`.

Значение по умолчанию: 1000000000 наносекунд (раз в секунду).

См. также:

- Системная таблица [trace\\_log](#)

## query\_profiler\_cpu\_time\_period\_ns

Устанавливает период для таймера CPU [query profiler](#). Этот таймер считает только время CPU.

Возможные значения:

- Положительное целое число в наносекундах.

Рекомендуемые значения:

- 10000000 (100 раз в секунду) наносекунд и большее значение для одиночных запросов.
- 1000000000 (раз в секунду) для профилирования в масштабе кластера.

- 0 для выключения таймера.

Тип: `UInt64`.

Значение по умолчанию: 1000000000 наносекунд.

См. также:

- Системная таблица [trace\\_log](#)

## allow\_introspection\_functions

Включает или отключает [функции самоанализа](#) для профилирования запросов.

Возможные значения:

- 1 — включены функции самоанализа.
- 0 — функции самоанализа отключены.

Значение по умолчанию: 0.

См. также

- [Sampling Query Profiler](#)
- Системная таблица [trace\\_log](#)

## input\_format\_parallel\_parsing

Включает или отключает режим, при котором входящие данные разбиваются на части, парсинг каждой из которых осуществляется параллельно с сохранением исходного порядка.

Поддерживается только для форматов [TSV](#), [TKSV](#), [CSV](#) и [JSONEachRow](#).

Возможные значения:

- 1 — включен режим параллельного разбора.
- 0 — отключен режим параллельного разбора.

Значение по умолчанию: 0.

## output\_format\_parallel\_formatting

Включает или отключает режим, при котором исходящие данные форматируются параллельно с сохранением исходного порядка. Поддерживается только для форматов [TSV](#), [TKSV](#), [CSV](#) и [JSONEachRow](#).

Возможные значения:

- 1 — включен режим параллельного форматирования.
- 0 — отключен режим параллельного форматирования.

Значение по умолчанию: 0.

## min\_chunk\_bytes\_for\_parallel\_parsing

- Тип: unsigned int
- Значение по умолчанию: 1 MiB

Минимальный размер блока в байтах, который каждый поток будет анализировать параллельно.

## output\_format\_avro\_codec

Устанавливает кодек сжатия, используемый для вывода файла Avro.

Тип: строка

Возможные значения:

- `null` — без сжатия
- `deflate` — сжать с помощью Deflate (zlib)
- `snappy` — сжать с помощью [Snappy](#)

Значение по умолчанию: `snappy` (если доступно) или `deflate`.

## output\_format\_avro\_sync\_interval

Устанавливает минимальный размер данных (в байтах) между маркерами синхронизации для выходного файла Avro.

Тип: unsigned int

Возможные значения: 32 (32 байта) - 1073741824 (1 GiB)

Значение по умолчанию: 32768 (32 KiB)

## background\_pool\_size

Задает количество потоков для выполнения фоновых операций в движках таблиц (например, слияния в таблицах с движком [MergeTree](#)). Настройка применяется при запуске сервера ClickHouse и не может быть изменена во пользовательском сеансе. Настройка позволяет управлять загрузкой процессора и диска. Чем меньше пул, тем ниже нагрузка на CPU и диск, при этом фоновые процессы работают с меньшей интенсивностью, что в конечном итоге может повлиять на производительность запросов, потому что сервер будет обрабатывать больше кусков.

Допустимые значения:

- Положительное целое число.

Значение по умолчанию: 16.

## merge\_selecting\_sleep\_ms

Время ожидания для слияния выборки, если ни один кусок не выбран. Снижение времени ожидания приводит к частому выбору задач в пуле `background_schedule_pool` и увеличению количества запросов к Zookeeper в крупных кластерах.

Возможные значения:

- Положительное целое число.

Значение по умолчанию: 5000.

## parallel\_distributed\_insert\_select

Включает параллельную обработку распределённых запросов `INSERT ... SELECT`.

Если при выполнении запроса `INSERT INTO distributed_table_a SELECT ... FROM distributed_table_b` оказывается, что обе таблицы находятся в одном кластере, то независимо от того [реплицируемые](#) они или нет, запрос выполняется локально на каждом шарде.

Допустимые значения:

- 0 — выключена.
- 1 — включена.

Значение по умолчанию: 0.

## insert\_distributed\_sync

Включает или отключает режим синхронного добавления данных в распределенные таблицы (таблицы с движком [Distributed](#)).

По умолчанию ClickHouse вставляет данные в распределённую таблицу в асинхронном режиме. Если `insert_distributed_sync=1`, то данные вставляются синхронно, а запрос `INSERT` считается выполненным успешно, когда данные записаны на все шарды (по крайней мере на одну реплику для каждого шарда, если `internal_replication = true`).

Возможные значения:

- 0 — Данные добавляются в асинхронном режиме.
- 1 — Данные добавляются в синхронном режиме.

Значение по умолчанию: 0.

### См. также

- [Движок Distributed](#)
- [Управление распределёнными таблицами](#)

## insert\_distributed\_one\_random\_shard

Включает или отключает режим вставки данных в [Distributed](#) таблицу в случайный шард при отсутствии ключа шардирования.

По умолчанию при вставке данных в `Distributed` таблицу с несколькими шардами и при отсутствии ключа шардирования сервер ClickHouse будет отклонять любой запрос на вставку данных. Когда `insert_distributed_one_random_shard = 1`, вставки принимаются, а данные записываются в случайный шард.

Возможные значения:

- 0 — если у таблицы несколько шардов, но ключ шардирования отсутствует, вставка данных отклоняется.
- 1 — если ключ шардирования отсутствует, то вставка данных осуществляется в случайный шард среди всех доступных шардов.

Значение по умолчанию: 0.

## insert\_shard\_id

Если не 0, указывает, в какой шард [Distributed](#) таблицы данные будут вставлены синхронно.

Если значение настройки `insert_shard_id` указано неверно, сервер выдаст ошибку.

Узнать количество шардов `shard_num` на кластере `requested_cluster` можно из конфигурации сервера, либо используя запрос:

```
SELECT uniq(shard_num) FROM system.clusters WHERE cluster = 'requested_cluster';
```

Возможные значения:

- 0 — выключено.
- Любое число от 1 до `shards_num` соответствующей [Distributed](#) таблицы.

Значение по умолчанию: 0.

## Пример

Запрос:

```
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;
```

Результат:

number
0
0
1
1
2
2
3
3
4
4

## validate\_polygons

Включает или отключает генерирование исключения в функции [pointInPolygon](#), если многоугольник самопересекающийся или самокасающийся.

Допустимые значения:

- 0 — генерирование исключения отключено. `pointInPolygon` принимает недопустимые многоугольники и возвращает для них, возможно, неверные результаты.
- 1 — генерирование исключения включено.

Значение по умолчанию: 1.

## always\_fetch\_merged\_part

Запрещает слияние данных для таблиц семейства [Replicated\\*MergeTree](#).

Если слияние запрещено, реплика никогда не выполняет слияние отдельных кусков данных, а всегда загружает объединённые данные из других реплик. Если объединённых данных пока нет, реплика ждет их появления. Нагрузка на процессор и диски на реплике уменьшается, но нагрузка на сеть в кластере возрастает. Настройка может быть полезна на репликах с относительно слабыми процессорами или медленными дисками, например, на репликах для хранения архивных данных.

Возможные значения:

- 0 — таблицы семейства `Replicated*MergeTree` выполняют слияние данных на реплике.

- 1 — таблицы семейства Replicated\*MergeTree не выполняют слияние данных на реплике, а загружают объединённые данные из других реплик.

Значение по умолчанию: 0.

#### См. также:

- [Репликация данных](#)

## transform\_null\_in

Разрешает сравнивать значения **NULL** в операторе **IN**.

По умолчанию, значения **NULL** нельзя сравнивать, поскольку **NULL** обозначает неопределенное значение. Следовательно, сравнение `expr = NULL` должно всегда возвращать `false`. С этой настройкой `NULL = NULL` возвращает `true` в операторе **IN**.

Possible values:

- 0 — Сравнение значений **NULL** в операторе **IN** возвращает `false`.
- 1 — Сравнение значений **NULL** в операторе **IN** возвращает `true`.

Значение по умолчанию: 0.

#### Пример

Рассмотрим таблицу `null_in`:

idx	i
1	1
2	NULL
3	3

Consider the `null_in` table:

idx	i
1	1
2	NULL
3	3

Запрос:

```
SELECT idx, i FROM null_in WHERE i IN (1, NULL) SETTINGS transform_null_in = 0;
```

Ответ:

idx	i
1	1

Запрос:

```
SELECT idx, i FROM null_in WHERE i IN (1, NULL) SETTINGS transform_null_in = 1;
```

Ответ:

idx	i
1	1
2	NULL

## См. также

- [Обработка значения NULL в операторе IN](#)

## low\_cardinality\_max\_dictionary\_size

Задает максимальный размер общего глобального словаря (в строках) для типа данных `LowCardinality`, который может быть записан в файловую систему хранилища. Настройка предотвращает проблемы с оперативной памятью в случае неограниченного увеличения словаря. Все данные, которые не могут быть закодированы из-за ограничения максимального размера словаря, ClickHouse записывает обычным способом.

Допустимые значения:

- Положительное целое число.

Значение по умолчанию: 8192.

## low\_cardinality\_use\_single\_dictionary\_for\_part

Включает или выключает использование единого словаря для куска (парта).

По умолчанию сервер ClickHouse следит за размером словарей, и если словарь переполняется, сервер создает следующий. Чтобы запретить создание нескольких словарей, задайте настройку `low_cardinality_use_single_dictionary_for_part = 1`.

Допустимые значения:

- 1 — Создание нескольких словарей для частей данных запрещено.
- 0 — Создание нескольких словарей для частей данных не запрещено.

Значение по умолчанию: 0.

## low\_cardinality\_allow\_in\_native\_format

Разрешает или запрещает использование типа данных `LowCardinality` с форматом данных `Native`.

Если использование типа `LowCardinality` ограничено, сервер ClickHouse преобразует столбцы `LowCardinality` в обычные столбцы для запросов `SELECT`, а обычные столбцы - в столбцы `LowCardinality` для запросов `INSERT`.

В основном настройка используется для сторонних клиентов, не поддерживающих тип данных `LowCardinality`.

Допустимые значения:

- 1 — Использование `LowCardinality` не ограничено.
- 0 — Использование `LowCardinality` ограничено.

Значение по умолчанию: 1.

## allow\_suspicious\_low\_cardinality\_types

Разрешает или запрещает использование типа данных `LowCardinality` с типами данных с фиксированным размером 8 байт или меньше: числовые типы данных и `FixedString(8_bytes_or_less)`.

Для небольших фиксированных значений использование `LowCardinality` обычно неэффективно, поскольку ClickHouse хранит числовой индекс для каждой строки. В результате:

- Используется больше дискового пространства.
- Потребление ОЗУ увеличивается, в зависимости от размера словаря.
- Некоторые функции работают медленнее из-за дополнительных операций кодирования.

Время слияния в таблицах на движке `MergeTree` также может увеличиться по описанным выше причинам.

Допустимые значения:

- 1 — Использование `LowCardinality` не ограничено.
- 0 — Использование `LowCardinality` ограничено.

Значение по умолчанию: 0.

## `background_buffer_flush_schedule_pool_size`

Задает количество потоков для выполнения фонового сброса данных в таблицах с движком `Buffer`. Настройка применяется при запуске сервера ClickHouse и не может быть изменена в пользовательском сеансе.

Допустимые значения:

- Положительное целое число.

Значение по умолчанию: 16.

## `background_move_pool_size`

Задает количество потоков для фоновых перемещений кусков между дисками. Работает для таблиц с движком `MergeTree`. Настройка применяется при запуске сервера ClickHouse и не может быть изменена в пользовательском сеансе.

Допустимые значения:

- Положительное целое число.

Значение по умолчанию: 8.

## `background_schedule_pool_size`

Задает количество потоков для выполнения фоновых задач. Работает для [реплицируемых](#) таблиц, стримов в [Kafka](#) и обновления IP адресов у записей во внутреннем [DNS кеше](#). Настройка применяется при запуске сервера ClickHouse и не может быть изменена в пользовательском сеансе.

Допустимые значения:

- Положительное целое число.

Значение по умолчанию: 128.

## `background_fetches_pool_size`

Задает количество потоков для скачивания кусков данных для [реплицируемых](#) таблиц. Настройка применяется при запуске сервера ClickHouse и не может быть изменена в пользовательском сеансе. Для использования в продакшне с частыми небольшими вставками или медленным кластером ZooKeeper рекомендуется использовать значение по умолчанию.

Допустимые значения:

- Положительное целое число.

Значение по умолчанию: 8.

## background\_distributed\_schedule\_pool\_size

Задает количество потоков для выполнения фоновых задач. Работает для таблиц с движком [Distributed](#). Настройка применяется при запуске сервера ClickHouse и не может быть изменена в пользовательском сеансе.

Допустимые значения:

- Положительное целое число.

Значение по умолчанию: 16.

## background\_message\_broker\_schedule\_pool\_size

Задает количество потоков для фонового потокового вывода сообщений. Настройка применяется при запуске сервера ClickHouse и не может быть изменена в пользовательском сеансе.

Допустимые значения:

- Положительное целое число.

Значение по умолчанию: 16.

### Смотрите также

- Движок [Kafka](#).
- Движок [RabbitMQ](#).

## format\_avro\_schema\_registry\_url

Задает URL реестра схем [Confluent](#) для использования с форматом [AvroConfluent](#).

Значение по умолчанию: Пустая строка.

## input\_format\_avro\_allow\_missing\_fields

Позволяет использовать данные, которых не нашлось в схеме формата [Avro](#) или [AvroConfluent](#). Если поле не найдено в схеме, ClickHouse подставит значение по умолчанию вместо исключения.

Возможные значения:

- 0 — Выключена.
- 1 — Включена.

Значение по умолчанию: 0.

## min\_insert\_block\_size\_rows\_for\_materialized\_views

Устанавливает минимальное количество строк в блоке, который может быть вставлен в таблицу запросом `INSERT`. Блоки меньшего размера склеиваются в блоки большего размера. Настройка применяется только для блоков, вставляемых в [материализованное представление](#). Настройка позволяет избежать избыточного потребления памяти.

Допустимые значения:

- Положительное целое число.
- 0 — Склейка блоков выключена.

Значение по умолчанию: 1048576.

**См. также:**

- [min\\_insert\\_block\\_size\\_rows](#)

## min\_insert\_block\_size\_bytes\_for\_materialized\_views

Устанавливает минимальное количество байтов в блоке, который может быть вставлен в таблицу запросом `INSERT`. Блоки меньшего размера склеиваются в блоки большего размера. Настройка применяется только для блоков, вставляемых в [материализованное представление](#). Настройка позволяет избежать избыточного потребления памяти.

Допустимые значения:

- Положительное целое число.
- 0 — Склейка блоков выключена.

Значение по умолчанию: 268435456.

**См. также:**

- [min\\_insert\\_block\\_size\\_bytes](#)

## output\_format\_pretty\_grid\_charset

Позволяет изменить кодировку, которая используется для отрисовки таблицы при выводе результатов запросов. Доступны следующие кодировки: UTF-8, ASCII.

### Пример

```
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

Включает или отключает оптимизацию в запросах `SELECT` с секцией `ORDER BY` при работе с таблицами семейства [MergeTree](#).

Возможные значения:

- 0 — оптимизация отключена.
- 1 — оптимизация включена.

Значение по умолчанию: 1.

#### **См. также**

- [Оптимизация чтения данных](#) в секции ORDER BY

## **optimize\_aggregation\_in\_order**

Включает или отключает оптимизацию в запросах **SELECT** с секцией **GROUP BY** при наличии подходящих ключей сортировки. Используется при работе с таблицами **MergeTree**.

Возможные значения:

- 0 — оптимизация по ключу сортировки отключена.
- 1 — оптимизация по ключу сортировки включена.

Значение по умолчанию: 0.

#### **См. также**

- [Оптимизация GROUP BY для отсортированных таблиц](#)

## **mutations\_sync**

Позволяет выполнять запросы **ALTER TABLE ... UPDATE|DELETE** ([мутации](#)) синхронно.

Возможные значения:

- 0 - мутации выполняются асинхронно.
- 1 - запрос ждет завершения всех мутаций на текущем сервере.
- 2 - запрос ждет завершения всех мутаций на всех репликах (если они есть).

Значение по умолчанию: 0.

#### **См. также**

- [Синхронность запросов ALTER](#)
- [Мутации](#)

## **ttl\_only\_drop\_parts**

Для таблиц **MergeTree** включает или отключает возможность полного удаления кусков данных, в которых все записи устарели.

Когда настройка **ttl\_only\_drop\_parts** отключена (т.е. по умолчанию), сервер лишь удаляет устаревшие записи в соответствии с их временем жизни (TTL).

Когда настройка **ttl\_only\_drop\_parts** включена, сервер целиком удаляет куски данных, в которых все записи устарели.

Удаление целых кусков данных вместо удаления отдельных записей позволяет устанавливать меньший таймаут **merge\_with\_ttl\_timeout** и уменьшает нагрузку на сервер, что способствует росту производительности.

Возможные значения:

- 0 — Возможность удаления целых кусков данных отключена.
- 1 — Возможность удаления целых кусков данных включена.

Значение по умолчанию: 0.

#### См. также

- Секции и настройки запроса **CREATE TABLE** (настройка `merge_with_ttl_timeout`)
- [Table TTL](#)

## output\_format\_pretty\_max\_value\_width

Ограничивает длину значения, выводимого в формате **Pretty**. Если значение длиннее указанного количества символов, оно обрезается.

Возможные значения:

- Положительное целое число.
- 0 — значение обрезается полностью.

Значение по умолчанию: 10000 символов.

#### Примеры

Запрос:

```
SET output_format_pretty_max_value_width = 10;
SELECT range(number) FROM system.numbers LIMIT 10 FORMAT PrettyCompactNoEscapes;
```

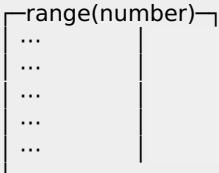
Результат:

```
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...
```

Запрос, где длина выводимого значения ограничена 0 символов:

```
SET output_format_pretty_max_value_width = 0;
SELECT range(number) FROM system.numbers LIMIT 5 FORMAT PrettyCompactNoEscapes;
```

Результат:



## output\_format\_pretty\_row\_numbers

Включает режим отображения номеров строк для запросов, выводимых в формате [Pretty](#).

Возможные значения:

- 0 — номера строк не выводятся.
- 1 — номера строк выводятся.

Значение по умолчанию: 0.

### Пример

Запрос:

```
SET output_format_pretty_row_numbers = 1;
SELECT TOP 3 name, value FROM system.settings;
```

Результат:

	name	value
1.	min_compress_block_size	65536
2.	max_compress_block_size	1048576
3.	max_block_size	65505

## system\_events\_show\_zero\_values

Позволяет выбрать события с нулевыми значениями из таблицы `system.events`.

В некоторые системы мониторинга вам нужно передать значения всех измерений (для каждой контрольной точки), даже если в результате — "0".

Возможные значения:

- 0 — настройка отключена — вы получите все события.
- 1 — настройка включена — вы сможете отсортировать события по нулевым и остальным значениям.

Значение по умолчанию: 0.

### Примеры

Запрос

```
SELECT * FROM system.events WHERE event='QueryMemoryLimitExceeded';
```

Результат

Ok.

## Запрос

```
SET system_events_show_zero_values = 1;
SELECT * FROM system.events WHERE event='QueryMemoryLimitExceeded';
```

## Результат

event	value	description
QueryMemoryLimitExceeded	0	Number of times when memory limit exceeded for query.

## lock\_acquire\_timeout

Устанавливает, сколько секунд сервер ожидает возможности выполнить блокировку таблицы.

Таймаут устанавливается для защиты от взаимоблокировки при выполнении операций чтения или записи. Если время ожидания истекло, а блокировку выполнить не удалось, сервер возвращает исключение с кодом `DEADLOCK_AVOIDED` и сообщением "Locking attempt timed out! Possible deadlock avoided. Client should retry." ("Время ожидания блокировки истекло! Возможная взаимоблокировка предотвращена. Повторите запрос.").

Возможные значения:

- Положительное целое число (в секундах).
- 0 — таймаут не устанавливается.

Значение по умолчанию: 120 секунд.

## cast\_keep\_nullable

Включает или отключает сохранение типа `Nullable` для аргумента функции `CAST`.

Если настройка включена, то когда в функцию `CAST` передается аргумент с типом `Nullable`, функция возвращает результат, также преобразованный к типу `Nullable`.

Если настройка отключена, то функция `CAST` всегда возвращает результат строго указанного типа.

Возможные значения:

- 0 — функция `CAST` преобразует аргумент строго к указанному типу.
- 1 — если аргумент имеет тип `Nullable`, то функция `CAST` преобразует его к типу `Nullable` для указанного типа.

Значение по умолчанию: 0.

## Примеры

Запрос возвращает аргумент, преобразованный строго к указанному типу:

```
SET cast_keep_nullable = 0;
SELECT CAST(toNullable(toInt32(0)) AS Int32) as x, toTypeName(x);
```

Результат:

```
x  toTypeName(CAST(toNullable(toInt32(0)), 'Int32'))  
0 | Int32
```

Запрос возвращает аргумент, преобразованный к типу `Nullable` для указанного типа:

```
SET cast_keep_nullable = 1;  
SELECT CAST(toNullable(toInt32(0)) AS Int32) as x, toTypeName(x);
```

Результат:

```
x  toTypeName(CAST(toNullable(toInt32(0)), 'Int32'))  
0 | Nullable(Int32)
```

## См. также

- Функция [CAST](#)

## persistent

Отключает перманентность для табличных движков [Set](#) и [Join](#).

Уменьшает расходы на ввод/вывод. Может быть полезно, когда требуется высокая производительность, а перманентность не обязательна.

Возможные значения:

- 1 — включено.
- 0 — отключено.

Значение по умолчанию: 1.

## output\_format\_csv\_null\_representation

Определяет представление `NUL` для формата выходных данных [CSV](#). Пользователь может установить в качестве значения любую строку, например, `My NULL`.

Значение по умолчанию: `\N`.

## Примеры

Запрос:

```
SELECT * FROM csv_custom_null FORMAT CSV;
```

Результат:

```
788  
\N  
\N
```

Запрос:

```
SET output_format_csv_null_representation = 'My NULL';  
SELECT * FROM csv_custom_null FORMAT CSV;
```

Результат:

```
788
My NULL
My NULL
```

## output\_format\_tsv\_null\_representation

Определяет представление `NUL` для формата выходных данных [TSV](#). Пользователь может установить в качестве значения любую строку.

Значение по умолчанию: `\N`.

### Примеры

Запрос

```
SELECT * FROM tsv_custom_null FORMAT TSV;
```

Результат

```
788
\N
\N
```

Запрос

```
SET output_format_tsv_null_representation = 'My NULL';
SELECT * FROM tsv_custom_null FORMAT TSV;
```

Результат

```
788
My NULL
My NULL
```

## output\_format\_json\_array\_of\_rows

Позволяет выводить все строки в виде массива JSON в формате [JSONEachRow](#).

Возможные значения:

- 1 — ClickHouse выводит все строки в виде массива и при этом каждую строку в формате [JSONEachRow](#).
- 0 — ClickHouse выводит каждую строку отдельно в формате [JSONEachRow](#).

Значение по умолчанию: 0.

### Пример запроса с включенной настройкой

Запрос:

```
SET output_format_json_array_of_rows = 1;
SELECT number FROM numbers(3) FORMAT JSONEachRow;
```

Результат:

```
[  
 {"number":"0"},  
 {"number":"1"},  
 {"number":"2"}  
]
```

## Пример запроса с отключенной настройкой

Запрос:

```
SET output_format_json_array_of_rows = 0;  
SELECT number FROM numbers(3) FORMAT JSONEachRow;
```

Результат:

```
{"number":"0"}  
 {"number":"1"}  
 {"number":"2"}
```

## allow\_nullable\_key

Включает или отключает поддержку типа `Nullable` для ключей таблиц `MergeTree`.

Возможные значения:

- 1 — включает поддержку типа `Nullable` для ключей таблиц.
- 0 — отключает поддержку типа `Nullable` для ключей таблиц.

Значение по умолчанию: 0.

## aggregate\_functions\_null\_for\_empty

Включает или отключает перезапись всех агрегатных функций в запросе, с добавлением к ним суффикса `-OrNull`. Включите для совместимости со стандартом SQL.

Реализуется с помощью перезаписи запросов (аналогично настройке `count_distinctImplementation`), чтобы получить согласованные результаты для распределенных запросов.

Возможные значения:

- 0 — выключена.
- 1 — включена.

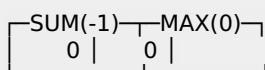
Значение по умолчанию: 0.

## Пример

Рассмотрим запрос с агрегирующими функциями:

```
SELECT SUM(-1), MAX(0) FROM system.one WHERE 0;
```

Результат запроса с настройкой `aggregate_functions_null_for_empty = 0`:



Результат запроса с настройкой `aggregate_functions_null_for_empty = 1`:

SUMOrNull(-1)	MAXOrNull(0)
NULL	NULL

## union\_default\_mode

Устанавливает режим объединения результатов `SELECT` запросов. Настройка используется только при совместном использовании с `UNION` без явного указания `UNION ALL` или `UNION DISTINCT`.

Возможные значения:

- `'DISTINCT'` — ClickHouse выводит строки в результате объединения результатов запросов, удаляя повторяющиеся строки.
- `'ALL'` — ClickHouse выводит все строки в результате объединения результатов запросов, включая повторяющиеся строки.
- `" "` — ClickHouse генерирует исключение при использовании с `UNION`.

Значение по умолчанию: `" "`.

Смотрите примеры в разделе [UNION](#).

## data\_type\_default\_nullable

Позволяет использовать по умолчанию тип данных `Nullable` в определении столбца без явных модификаторов `NONE` или `NOT NULL`.

Возможные значения:

- 1 — типы данных в определении столбца заданы по умолчанию как `Nullable`.
- 0 — типы данных в определении столбца не заданы по умолчанию как `Nullable`.

Значение по умолчанию: 0.

## execute\_merges\_on\_single\_replica\_time\_threshold

Включает особую логику выполнения слияний на репликах.

Возможные значения:

- Положительное целое число (в секундах).
- 0 — не используется особая логика выполнения слияний. Слияния происходят обычным образом на всех репликах.

Значение по умолчанию: 0.

## Использование

Выбирается одна реплика для выполнения слияния. Устанавливается порог времени с момента начала слияния. Другие реплики ждут завершения слияния, а затем скачивают результат. Если время выполнения слияния превышает установленный порог и выбранная реплика не выполняет слияние, тогда слияние выполняется на других репликах как обычно.

Большие значения этой настройки могут привести к задержкам репликации.

Эта настройка полезна, когда скорость слияния ограничивается мощностью процессора, а не скоростью операций ввода-вывода (при выполнении "тяжелого" сжатия данных, при расчете агрегатных функций или выражений по умолчанию, требующих большого объема вычислений, или просто при большом количестве мелких слияний).

## max\_final\_threads

Устанавливает максимальное количество параллельных потоков для фазы чтения данных запроса `SELECT` с модификатором **FINAL**.

Возможные значения:

- Положительное целое число.
- 0 или 1 — настройка отключена. `SELECT` запросы выполняются в один поток.

Значение по умолчанию: 16.

## opentelemetry\_start\_trace\_probability

Задает вероятность того, что ClickHouse начнет трассировку для выполненных запросов (если не указан **входящий контекст** трассировки).

Возможные значения:

- 0 — трассировка для выполненных запросов отключена (если не указан входящий контекст трассировки).
- Положительное число с плавающей точкой в диапазоне [0..1]. Например, при значении настройки, равной 0,5, ClickHouse начнет трассировку в среднем для половины запросов.
- 1 — трассировка для всех выполненных запросов включена.

Значение по умолчанию: 0.

## optimize\_on\_insert

Включает или выключает преобразование данных перед добавлением в таблицу, как будто над добавляемым блоком предварительно было произведено слияние (в соответствии с движком таблицы).

Возможные значения:

- 0 — выключена
- 1 — включена.

Значение по умолчанию: 1.

### Пример

Сравните добавление данных при включенной и выключенной настройке:

Запрос:

```
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;
```

Результат:

FirstTable
0
1

SecondTable
0
0
0
1
1

Обратите внимание на то, что эта настройка влияет на поведение [материализованных представлений](#) и БД [MaterializedMySQL](#).

## engine\_file\_empty\_if\_not\_exists

Включает или отключает возможность выполнять запрос `SELECT` к таблице на движке [File](#), не содержащей файл.

Возможные значения:

- 0 — запрос `SELECT` генерирует исключение.
- 1 — запрос `SELECT` возвращает пустой результат.

Значение по умолчанию: 0.

## engine\_file\_truncate\_on\_insert

Включает или выключает удаление данных из таблицы до вставки в таблицу на движке [File](#).

Возможные значения:

- 0 — запрос `INSERT` добавляет данные в конец файла после существующих.
- 1 — `INSERT` удаляет имеющиеся в файле данные и замещает их новыми.

Значение по умолчанию: 0.

## allow\_experimental\_geo\_types

Разрешает использование экспериментальных типов данных для работы с [географическими структурами](#).

Возможные значения:

- 0 — использование типов данных для работы с географическими структурами не поддерживается.
- 1 — использование типов данных для работы с географическими структурами поддерживается.

Значение по умолчанию: 0.

## database\_atomic\_wait\_for\_drop\_and\_detach\_synchronously

Добавляет модификатор `SYNC` ко всем запросам `DROP` и `DETACH`.

Возможные значения:

- 0 — Запросы будут выполняться с задержкой.
- 1 — Запросы будут выполняться без задержки.

Значение по умолчанию: 0.

## show\_table\_uuid\_in\_table\_create\_query\_if\_not\_nil

Устанавливает отображение запроса `SHOW TABLE`.

Возможные значения:

- 0 — Запрос будет отображаться без UUID таблицы.
- 1 — Запрос будет отображаться с UUID таблицы.

Значение по умолчанию: 0.

## allow\_experimental\_live\_view

Включает экспериментальную возможность использования [LIVE-представлений](#).

Возможные значения:

- 0 — живые представления не поддерживаются.
- 1 — живые представления поддерживаются.

Значение по умолчанию: 0.

## live\_view\_heartbeat\_interval

Задает интервал в секундах для периодической проверки существования [LIVE VIEW](#).

Значение по умолчанию: 15.

## max\_live\_view\_insert\_blocks\_before\_refresh

Задает наибольшее число вставок, после которых запрос на формирование [LIVE VIEW](#) исполняется снова.

Значение по умолчанию: 64.

## temporary\_live\_view\_timeout

Задает время в секундах, после которого [LIVE VIEW](#) удаляется.

Значение по умолчанию: 5.

## periodic\_live\_view\_refresh

Задает время в секундах, по истечении которого [LIVE VIEW](#) с установленным автообновлением обновляется.

Значение по умолчанию: 60.

## check\_query\_single\_value\_result

Определяет уровень детализации результата для запросов [CHECK TABLE](#) для таблиц семейства MergeTree.

Возможные значения:

- 0 — запрос возвращает статус каждого куска данных таблицы.
- 1 — запрос возвращает статус таблицы в целом.

Значение по умолчанию: 0.

## prefer\_column\_name\_to\_alias

Включает или отключает замену названий столбцов на псевдонимы (alias) в выражениях и секциях запросов, см. [Примечания по использованию синонимов](#). Включите эту настройку, чтобы синтаксис псевдонимов в ClickHouse был более совместим с большинством других СУБД.

Возможные значения:

- 0 — псевдоним подставляется вместо имени столбца.
- 1 — псевдоним не подставляется вместо имени столбца.

Значение по умолчанию: 0.

### Пример

Какие изменения привносит включение и выключение настройки:

Запрос:

```
SET prefer_column_name_to_alias = 0;
SELECT avg(number) AS number, max(number) FROM numbers(10);
```

Результат:

```
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.
```

Запрос:

```
SET prefer_column_name_to_alias = 1;
SELECT avg(number) AS number, max(number) FROM numbers(10);
```

Результат:

number	max(number)
4.5	9

## limit

Устанавливает максимальное количество строк, возвращаемых запросом. Ограничивает сверху значение, установленное в запросе в секции [LIMIT](#).

Возможные значения:

- 0 — число строк не ограничено.
- Положительное целое число.

Значение по умолчанию: 0.

## offset

Устанавливает количество строк, которые необходимо пропустить перед началом возврата строк из запроса. Суммируется со значением, установленным в запросе в секции **OFFSET**.

Возможные значения:

- 0 — строки не пропускаются.
- Положительное целое число.

Значение по умолчанию: 0.

## Пример

Исходная таблица:

```
CREATE TABLE test (i UInt64) ENGINE = MergeTree() ORDER BY i;
INSERT INTO test SELECT number FROM numbers(500);
```

Запрос:

```
SET limit = 5;
SET offset = 7;
SELECT * FROM test LIMIT 10 OFFSET 100;
```

Результат:

i
107
108
109

## http\_connection\_timeout

Тайм-аут для HTTP-соединения (в секундах).

Возможные значения:

- 0 - бесконечный тайм-аут.
- Любое положительное целое число.

Значение по умолчанию: 1.

## http\_send\_timeout

Тайм-аут для отправки данных через HTTP-интерфейс (в секундах).

Возможные значения:

- 0 - бесконечный тайм-аут.
- Любое положительное целое число.

Значение по умолчанию: 1800.

## http\_receive\_timeout

Тайм-аут для получения данных через HTTP-интерфейс (в секундах).

Возможные значения:

- 0 - бесконечный тайм-аут.
- Любое положительное целое число.

Значение по умолчанию: 1800.

## optimize\_syntax\_fuse\_functions

Позволяет объединить агрегатные функции с одинаковым аргументом. Запрос, содержащий по крайней мере две агрегатные функции: `sum`, `count` или `avg` с одинаковым аргументом, перезаписывается как `sumCount`.

Возможные значения:

- 0 — функции с одинаковым аргументом не объединяются.
- 1 — функции с одинаковым аргументом объединяются.

Значение по умолчанию: 0.

### Пример

Запрос:

```
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;
```

Результат:

```
SELECT
    sum(a),
    sumCount(b).1,
    sumCount(b).2,
    (sumCount(b).1) / (sumCount(b).2)
FROM fuse_tbl
```

## allow\_experimental\_database\_replicated

Позволяет создавать базы данных с движком `Replicated`.

Возможные значения:

- 0 — Disabled.
- 1 — Enabled.

Значение по умолчанию: 0.

## database\_replicated\_initial\_query\_timeout\_sec

Устанавливает, как долго начальный DDL-запрос должен ждать, пока реплицированная база данных прецессирует предыдущие записи очереди DDL в секундах.

Возможные значения:

- Положительное целое число.
- 0 — Не ограничено.

Значение по умолчанию: 300.

## distributed\_ddl\_task\_timeout

Устанавливает тайм-аут для ответов на DDL-запросы от всех хостов в кластере. Если DDL-запрос не был выполнен на всех хостах, ответ будет содержать ошибку тайм-аута, и запрос будет выполнен в асинхронном режиме.

Возможные значения:

- Положительное целое число.
- 0 — Асинхронный режим.
- Отрицательное число — бесконечный тайм-аут.

Значение по умолчанию: 180.

## distributed\_ddl\_output\_mode

Задает формат результата распределенного DDL-запроса.

Возможные значения:

- `throw` — возвращает набор результатов со статусом выполнения запросов для всех хостов, где завершен запрос. Если запрос не выполнился на некоторых хостах, то будет выброшено исключение. Если запрос еще не закончен на некоторых хостах и таймаут `distributed_ddl_task_timeout` превышен, то выбрасывается исключение `TIMEOUT_EXCEEDED`.
- `none` — идентично `throw`, но распределенный DDL-запрос не возвращает набор результатов.
- `null_status_on_timeout` — возвращает `NULL` в качестве статуса выполнения в некоторых строках набора результатов вместо выбрасывания `TIMEOUT_EXCEEDED`, если запрос не закончен на соответствующих хостах.
- `never_throw` — не выбрасывает исключение и `TIMEOUT_EXCEEDED`, если запрос не удался на некоторых хостах.

Значение по умолчанию: `throw`.

## flatten\_nested

Устанавливает формат данных у **вложенных** столбцов.

Возможные значения:

- 1 — вложенный столбец преобразуется к отдельным массивам.
- 0 — вложенный столбец преобразуется к массиву кортежей.

Значение по умолчанию: 1.

## Использование

Если установлено значение 0, можно использовать любой уровень вложенности.

## Примеры

Запрос:

```
SET flatten_nested = 1;  
  
CREATE TABLE t_nest (`n` Nested(a UInt32, b UInt32)) ENGINE = MergeTree ORDER BY tuple();  
  
SHOW CREATE TABLE t_nest;
```

Результат:

```
└─statement  
   └─CREATE TABLE default.t_nest  
     ( `n.a` Array(UInt32),  
       `n.b` Array(UInt32)  
     )  
     ENGINE = MergeTree  
     ORDER BY tuple()  
     SETTINGS index_granularity = 8192 |
```

Запрос:

```
SET flatten_nested = 0;  
  
CREATE TABLE t_nest (`n` Nested(a UInt32, b UInt32)) ENGINE = MergeTree ORDER BY tuple();  
  
SHOW CREATE TABLE t_nest;
```

Результат:

```
└─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

Определяет, как табличные функции [mysql](#), [postgresql](#) и [odbc](#)] используют Nullable столбцы.

Возможные значения:

- 0 — табличная функция явно использует Nullable столбцы.
- 1 — табличная функция неявно использует Nullable столбцы.

Значение по умолчанию: 1.

## Использование

Если установлено значение 0, то табличная функция не делает Nullable столбцы, а вместо NULL выставляет значения по умолчанию для скалярного типа. Это также применимо для значений NULL внутри массивов.

## output\_format\_arrow\_low\_cardinality\_as\_dictionary

Позволяет конвертировать тип `LowCardinality` в тип `DICTIONARY` формата `Arrow` для запросов `SELECT`.

Возможные значения:

- 0 — тип `LowCardinality` не конвертируется в тип `DICTIONARY`.
- 1 — тип `LowCardinality` конвертируется в тип `DICTIONARY`.

Значение по умолчанию: 0.

## materialized\_postgresql\_max\_block\_size

Задает максимальное количество строк, собранных в памяти перед вставкой данных в таблицу базы данных PostgreSQL.

Возможные значения:

- Положительное целое число.

Значение по умолчанию: 65536.

## materialized\_postgresql\_tables\_list

Задает список таблиц базы данных PostgreSQL, разделенных запятыми, которые будут реплицироваться с помощью движка базы данных `MaterializedPostgreSQL`.

Значение по умолчанию: пустой список — база данных PostgreSQL будет полностью реплицирована.

## materialized\_postgresql\_allow\_automatic\_update

Позволяет автоматически обновить таблицу в фоновом режиме при обнаружении изменений схемы. DDL-запросы на стороне сервера PostgreSQL не реплицируются с помощью движка ClickHouse `MaterializedPostgreSQL`, поскольку это запрещено протоколом логической репликации PostgreSQL, но факт DDL-изменений обнаруживается транзакционно. После обнаружения DDL по умолчанию прекращается репликация этих таблиц. Однако, если эта настройка включена, то вместо остановки репликации, таблицы будут перезагружены в фоновом режиме с помощью снимка базы данных без потери информации, и репликация для них будет продолжена.

Возможные значения:

- 0 — таблица не обновляется автоматически в фоновом режиме при обнаружении изменений схемы.
- 1 — таблица обновляется автоматически в фоновом режиме при обнаружении изменений схемы.

Значение по умолчанию: 0.

## materialized\_postgresql\_replication\_slot

Строка с идентификатором слота репликации, созданного пользователем вручную. Эта настройка должна использоваться совместно с `materialized_postgresql_snapshot`.

## `materialized_postgresql_snapshot`

Строка с идентификатором снэпшота, из которого будет выполняться [исходный дамп таблиц PostgreSQL](#). Эта настройка должна использоваться совместно с `materialized_postgresql_replication_slot`.

## `allow_experimental_projection_optimization`

Включает или отключает поддержку [проекций](#) при обработке запросов `SELECT`.

Возможные значения:

- 0 — Проекции не поддерживаются.
- 1 — Проекции поддерживаются.

Значение по умолчанию: 0.

## `force_optimize_projection`

Включает или отключает обязательное использование [проекций](#) в запросах `SELECT`, если поддержка проекций включена (см. настройку `allow_experimental_projection_optimization`).

Возможные значения:

- 0 — Проекции используются опционально.
- 1 — Проекции обязательно используются.

Значение по умолчанию: 0.

## `replication_alter_partitions_sync`

Позволяет настроить ожидание выполнения действий на репликах запросами `ALTER`, `OPTIMIZE` или `TRUNCATE`.

Возможные значения:

- 0 — не ждать.
- 1 — ждать выполнения действий на своей реплике.
- 2 — ждать выполнения действий на всех репликах.

Значение по умолчанию: 1.

## `replication_wait_for_inactive_replica_timeout`

Указывает время ожидания (в секундах) выполнения запросов `ALTER`, `OPTIMIZE` или `TRUNCATE` для неактивных реплик.

Возможные значения:

- 0 — не ждать.
- Отрицательное целое число — ждать неограниченное время.
- Положительное целое число — установить соответствующее количество секунд ожидания.

Значение по умолчанию: 120 секунд.

## `regexp_max_matches_per_row`

Задает максимальное количество совпадений для регулярного выражения. Настройка применяется для защиты памяти от перегрузки при использовании "жадных" квантификаторов в регулярном выражении для функции `extractAllGroupsHorizontal`.

Возможные значения:

- Положительное целое число.

Значение по умолчанию: 1000.

## http\_max\_single\_read\_retries

Задает максимальное количество попыток чтения данных во время одного HTTP-запроса.

Возможные значения:

- Положительное целое число.

Значение по умолчанию: 1024.

## log\_queries\_probability

Позволяет пользователю записывать в системные таблицы `query_log`, `query_thread_log` и `query_views_log` только часть запросов, выбранных случайным образом, с указанной вероятностью. Это помогает снизить нагрузку при большом объеме запросов в секунду.

Возможные значения:

- 0 — запросы не регистрируются в системных таблицах.
- Положительное число с плавающей точкой в диапазоне [0..1]. Например, при значении настройки, равном 0.5, примерно половина запросов регистрируется в системных таблицах.
- 1 — все запросы регистрируются в системных таблицах.

Значение по умолчанию: 1.

## short\_circuit\_function\_evaluation

Позволяет вычислять функции `if`, `multilf`, `and` и `or` по **короткой схеме**. Это помогает оптимизировать выполнение сложных выражений в этих функциях и предотвратить возможные исключения (например, деление на ноль, когда оно не ожидается).

Возможные значения:

- `enable` — по короткой схеме вычисляются функции, которые подходят для этого (могут сгенерировать исключение или требуют сложных вычислений).
- `force_enable` — все функции вычисляются по короткой схеме.
- `disable` — вычисление функций по короткой схеме отключено.

Значение по умолчанию: `enable`.

## max\_hyperscan\_regexp\_length

Задает максимальную длину каждого регулярного выражения в **hyperscan-функциях** поиска множественных совпадений в строке.

Возможные значения:

- Положительное целое число.

- 0 - длина не ограничена.

Значение по умолчанию: 0.

## Пример

Запрос:

```
SELECT multiMatchAny('abcd', ['ab','bcd','c','d']) SETTINGS max_hyperscan_regexp_length = 3;
```

Результат:

```
multiMatchAny('abcd', ['ab', 'bcd', 'c', 'd'])  
1 |
```

Запрос:

```
SELECT multiMatchAny('abcd', ['ab','bcd','c','d']) SETTINGS max_hyperscan_regexp_length = 2;
```

Результат:

```
Exception: Regexp length too large.
```

## См. также

- [max\\_hyperscan\\_regexp\\_total\\_length](#)

## max\_hyperscan\_regexp\_total\_length

Задает максимальную общую длину всех регулярных выражений в каждой [hyperscan-функции](#) поиска множественных совпадений в строке.

Возможные значения:

- Положительное целое число.
- 0 - длина не ограничена.

Значение по умолчанию: 0.

## Пример

Запрос:

```
SELECT multiMatchAny('abcd', ['a','b','c','d']) SETTINGS max_hyperscan_regexp_total_length = 5;
```

Результат:

```
multiMatchAny('abcd', ['a', 'b', 'c', 'd'])  
1 |
```

Запрос:

```
SELECT multiMatchAny('abcd', ['ab','bc','c','d']) SETTINGS max_hyperscan_regexp_total_length = 5;
```

Результат:

```
Exception: Total regexp lengths too large.
```

## См. также

- [max\\_hyperscan\\_regexp\\_length](#)

## enable\_positional\_arguments

Включает и отключает поддержку позиционных аргументов для **GROUP BY**, **LIMIT BY**, **ORDER BY**. Если вы хотите использовать номера столбцов вместо названий в выражениях этих операторов, установите `enable_positional_arguments = 1`.

Возможные значения:

- 0 — Позиционные аргументы не поддерживаются.
- 1 — Позиционные аргументы поддерживаются: можно использовать номера столбцов вместо названий столбцов.

Значение по умолчанию: 0.

## Пример

Запрос:

```
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;
```

Результат:

one	two	three
30	10	20
20	20	10
10	20	30

## optimize\_move\_to\_prewhere

Включает или отключает автоматическую оптимизацию **PREWHERE** в запросах **SELECT**.

Работает только с таблицами семейства [\\*MergeTree](#).

Возможные значения:

- 0 — автоматическая оптимизация **PREWHERE** отключена.
- 1 — автоматическая оптимизация **PREWHERE** включена.

Значение по умолчанию: 1.

## optimize\_move\_to\_prewhere\_if\_final

Включает или отключает автоматическую оптимизацию `PREWHERE` в запросах `SELECT` с модификатором `FINAL`.

Работает только с таблицами семейства `*MergeTree`.

Возможные значения:

- 0 — автоматическая оптимизация `PREWHERE` в запросах `SELECT` с модификатором `FINAL` отключена.
- 1 — автоматическая оптимизация `PREWHERE` в запросах `SELECT` с модификатором `FINAL` включена.

Значение по умолчанию: 0.

### См. также

- настройка `optimize_move_to_prewhere`

## describe\_include\_subcolumns

Включает или отключает описание подстолбцов при выполнении запроса `DESCRIBE`. Настройка действует, например, на элементы `Tuple` или подстолбцы типов `Map`, `Nullable` или `Array`.

Возможные значения:

- 0 — подстолбцы не включаются в результат запросов `DESCRIBE`.
- 1 — подстолбцы включаются в результат запросов `DESCRIBE`.

Значение по умолчанию: 0.

### Пример

Смотрите пример запроса `DESCRIBE`.

## async\_insert

Включает или отключает асинхронные вставки. Работает только для вставок по протоколу HTTP. Обратите внимание, что при таких вставках дедупликация не производится.

Если включено, данные собираются в пачки перед вставкой в таблицу. Это позволяет производить мелкие и частые вставки в ClickHouse (до 15000 запросов в секунду) без промежуточных таблиц.

Вставка данных происходит либо как только объем вставляемых данных превышает `async_insert_max_data_size`, либо через `async_insert_busy_timeout_ms` миллисекунд после первого запроса `INSERT`. Если в `async_insert_stale_timeout_ms` задано ненулевое значение, то данные вставляются через `async_insert_stale_timeout_ms` миллисекунд после последнего запроса.

Если включен параметр `wait_for_async_insert`, каждый клиент ждет, пока данные будут сброшены в таблицу. Иначе запрос будет обработан почти моментально, даже если данные еще не вставлены.

Возможные значения:

- 0 — вставки производятся синхронно, один запрос за другим.
- 1 — включены множественные асинхронные вставки.

Значение по умолчанию: 0.

## async\_insert\_threads

Максимальное число потоков для фоновой обработки и вставки данных.

Возможные значения:

- Положительное целое число.
- 0 — асинхронные вставки отключены.

Значение по умолчанию: 16.

## wait\_for\_async\_insert

Включает или отключает ожидание обработки асинхронных вставок. Если включено, клиент выведет OK только после того, как данные вставлены. Иначе будет выведен OK, даже если вставка не произошла.

Возможные значения:

- 0 — сервер возвращает OK даже если вставка данных еще не завершена.
- 1 — сервер возвращает OK только после завершения вставки данных.

Значение по умолчанию: 1.

## wait\_for\_async\_insert\_timeout

Время ожидания в секундах, выделяемое для обработки асинхронной вставки.

Возможные значения:

- Положительное целое число.
- 0 — ожидание отключено.

Значение по умолчанию: [lock\\_acquire\\_timeout](#).

## async\_insert\_max\_data\_size

Максимальный размер необработанных данных (в байтах), собранных за запрос, перед их вставкой.

Возможные значения:

- Положительное целое число.
- 0 — асинхронные вставки отключены.

Значение по умолчанию: 1000000.

## async\_insert\_busy\_timeout\_ms

Максимальное время ожидания в миллисекундах после первого запроса INSERT и перед вставкой данных.

Возможные значения:

- Положительное целое число.
- 0 — ожидание отключено.

Значение по умолчанию: 200.

## async\_insert\_stale\_timeout\_ms

Максимальное время ожидания в миллисекундах после последнего запроса `INSERT` и перед вставкой данных. Если установлено ненулевое значение, `async_insert_busy_timeout_ms` будет продлеваться с каждым запросом `INSERT`, пока не будет превышен `async_insert_max_data_size`.

Возможные значения:

- Положительное целое число.
- 0 — ожидание отключено.

Значение по умолчанию: 0.

## Профили настроек

Профиль настроек — это набор настроек, сгруппированных под одним именем.

### Информация

Для управления профилями настроек рекомендуется использовать **SQL-ориентированный воркфлоу**, который также поддерживается в ClickHouse.

Название профиля может быть любым. Вы можете указать один и тот же профиль для разных пользователей. Самое важное, что можно прописать в профиле — `readonly=1`, это обеспечит доступ только на чтение.

Профили настроек поддерживают наследование. Это реализуется указанием одной или нескольких настроек `profile` перед остальными настройками, перечисленными в профиле. Если одна настройка указана в нескольких профилях, используется последнее из значений.

Все настройки профиля можно применить, установив настройку `profile`.

Пример:

Установить профиль `web`.

```
SET profile = 'web'
```

Профили настроек объявляются в конфигурационном файле пользователей. Обычно это `users.xml`.

Пример:

```
<!-- 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>
```

В примере задано два профиля: `default` и `web`.

Профиль `default` имеет специальное значение — он обязательен и применяется при запуске сервера. Профиль `default` содержит настройки по умолчанию.

Профиль `web` — обычный профиль, который может быть установлен с помощью запроса `SET` или параметра URL при запросе по HTTP.

## Ограничения на изменение настроек

Ограничения на изменение настроек могут находиться внутри секции `profiles` файла `user.xml` и запрещают пользователю менять некоторые настройки с помощью запроса `SET`.

Выглядит это следующим образом:

```
<profiles>
  <имя_пользователя>
    <constraints>
      <настройка_1>
        <min>нижняя_граница</min>
      </настройка_1>
      <настройка_2>
        <max>верхняя_граница</max>
      </настройка_2>
      <настройка_3>
        <min>нижняя_граница</min>
        <max>верхняя_граница</max>
      </настройка_3>
      <настройка_4>
        <readonly/>
      </настройка_4>
    </constraints>
  </имя_пользователя>
</profiles>
```

Если пользователь пытается выйти за пределы, установленные этими ограничениями, то кидается исключение и настройка сохраняет прежнее значение.

Поддерживаются три типа ограничений: `min`, `max` и `readonly`. Ограничения `min` и `max` указывают нижнюю и верхнюю границы для числовых настроек и могут использоваться вместе.

Ограничение `readonly` указывает, что пользователь не может менять настройку.

**Пример:** Пусть файл `users.xml` содержит строки:

```
<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>
```

Каждый из следующих запросов кинет исключение:

```
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.
```

**Примечание:** профиль с именем `default` обрабатывается специальным образом: все ограничения на изменение настроек из этого профиля становятся дефолтными и влияют на всех пользователей, кроме тех, где эти ограничения явно переопределены.

## Настройки пользователей

Раздел `users` конфигурационного файла `user.xml` содержит настройки для пользователей.

# Информация

Для управления пользователями рекомендуется использовать **SQL-ориентированный воркфлоу**, который также поддерживается в ClickHouse.

Структура раздела users:

```
<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

Пароль можно указать в текстовом виде или в виде SHA256 (шестнадцатеричный формат).

- Чтобы назначить пароль в текстовом виде (**не рекомендуем**), поместите его в элемент password.

Например, `<password>qwerty</password>`. Пароль можно оставить пустым.

- Чтобы назначить пароль в виде SHA256, поместите хэш в элемент password\_sha256\_hex.

Например,  
`<password\_sha256\_hex>65e84be33532fb784c48129675f9eff3a682b27168c0ea744b2cf58ee02337c5</password\_sha256\_hex>`.

Пример создания пароля в командной строке:

```

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

Первая строка результата — пароль. Вторая строка — соответствующий ему хэш SHA256.

- Для совместимости с клиентами MySQL, пароль можно задать с помощью двойного хэша SHA1, поместив его в элемент `password_double_sha1_hex`.

Например,  
`<password_double_sha1_hex>08b4a0f1de6ad37da17359e592c8d74788a83eb0</password_double_sha1_hex>`.

Пример создания пароля в командной строке:

```
...
PASSWORD=$(base64 < /dev/urandom | head -c8); echo "$PASSWORD"; echo -n "$PASSWORD" | shasum | tr -d '-' | xxd -r -p | shasum | tr -d '-'
```

Первая строка результата — пароль. Вторая строка — соответствующий ему двойной хэш SHA1.

## access\_management

Включает или выключает SQL-ориентированное управление доступом для пользователя.

Возможные значения:

- 0 — Выключено.
- 1 — Включено.

Значение по умолчанию: 0.

## user\_name/networks

Список сетей, из которых пользователь может подключиться к серверу ClickHouse.

Каждый элемент списка имеет одну из следующих форм:

- `<ip>` — IP-адрес или маска подсети.

Примеры: `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>` — Имя хоста.

Пример: `example01.host.ru`.

Для проверки доступа выполняется DNS-запрос, и все возвращенные IP-адреса сравниваются с адресом клиента.

- `<host_regexp>` — Регулярное выражение для имен хостов.

Пример, `^example\d\d-\d\d\d\.\host\.ru\$`

Для проверки доступа выполняется [DNS запрос PTR](https://en.wikipedia.org/wiki/Reverse\_DNS\_lookup) для адреса клиента, а затем применяется заданное регулярное выражение. Затем, для результатов запроса PTR выполняется другой DNS-запрос и все полученные адреса сравниваются с адресом клиента.  
Рекомендуем завершать регулярное выражение символом \$.

Все результаты DNS-запросов кэшируются до перезапуска сервера.

## Примеры

Чтобы открыть доступ пользователю из любой сети, укажите:

```
<ip>::/0</ip>
```

## Внимание

Открывать доступ из любой сети небезопасно, если у вас нет правильно настроенного брандмауэра или сервер не отключен от интернета.

Чтобы открыть только локальный доступ, укажите:

```
<ip>::1</ip>
<ip>127.0.0.1</ip>
```

### user\_name/profile

Пользователю можно назначить профиль настроек. Профили настроек конфигурируются в отдельной секции файла `users.xml`. Подробнее читайте в разделе [Профили настроек](#).

### user\_name/quota

Квотирование позволяет отслеживать или ограничивать использование ресурсов в течение определённого периода времени. Квоты настраиваются в разделе `quotas` конфигурационного файла `users.xml`.

Пользователю можно назначить квоты. Подробное описание настройки квот смотрите в разделе [Квоты](#).

### user\_name/databases

В этом разделе вы можете ограничить выдачу ClickHouse запросами `SELECT` для конкретного пользователя, таким образом реализуя базовую защиту на уровне строк.

#### Пример

Следующая конфигурация задаёт, что пользователь `user1` в результате запросов `SELECT` может получать только те строки `table1`, в которых значение поля `id` равно 1000.

```
<user1>
  <databases>
    <database_name>
      <table1>
        <filter>id = 1000</filter>
      </table1>
    </database_name>
  </databases>
</user1>
```

Элемент `filter` содержит любое выражение, возвращающее значение типа `UInt8`. Обычно он содержит сравнения и логические операторы. Строки `database_name.table1`, для которых фильтр возвращает 0 не выдаются пользователю. Фильтрация несовместима с операциями `PREWHERE` и отключает оптимизацию `WHERE-PREWHERE`.

## Настройки MergeTree таблиц

Значения настроек всех MergeTree таблиц собраны в таблице `system.merge_tree_settings`. Их можно переопределить в разделе `merge_tree` файла `config.xml` или задать в секции `SETTINGS` каждой таблицы.

Пример переопределения в `config.xml`:

```
<merge_tree>
  <max_suspicious_broken_parts>5</max_suspicious_broken_parts>
</merge_tree>
```

Пример установки `SETTINGS` для конкретной таблицы:

```
CREATE TABLE foo
(
    `A` Int64
)
ENGINE = MergeTree
ORDER BY tuple()
SETTINGS max_suspicious_broken_parts = 500;
```

Пример изменения настроек для конкретной таблицы при помощи команды `ALTER TABLE ... MODIFY SETTING`:

```
ALTER TABLE foo
MODIFY SETTING max_suspicious_broken_parts = 100;
```

## parts\_to\_throw\_insert

Если число активных кусков в партиции больше значения `parts_to_throw_insert`, то `INSERT` прерывается с исключением: `Too many parts (N). Merges are processing significantly slower than inserts`

Возможные значения:

- Положительное целое число.

Значение по умолчанию: 300.

Чтобы производительность запросов `SELECT` стала максимальной, необходимо минимизировать количество обрабатываемых кусков, см. [Дизайн MergeTree](#).

Можно установить значение больше — 600 (1200) кусков. Тогда ошибка `Too many parts` будет появляться реже, но при этом могут возникнуть проблемы с фоновыми слияниями и производительностью `SELECT`-запросов.

## parts\_to\_delay\_insert

Если число кусков в партиции больше значения `parts_to_delay_insert`, то `INSERT` искусственно замедляется.

Возможные значения:

- Положительное целое число.

Значение по умолчанию: 150.

ClickHouse искусственно выполняет `INSERT` дольше (добавляет 'sleep') так, чтобы куски сливались в фоновом процессе быстрее, чем добавляются.

## inactive\_parts\_to\_throw\_insert

Если число неактивных кусков в партиции больше значения `inactive_parts_to_throw_insert`, то `INSERT` прерывается с исключением `Too many inactive parts (N). Parts cleaning are processing significantly slower than inserts`.

Возможные значения:

- Положительное целое число.

Значение по умолчанию: 0 (без ограничений).

## inactive\_parts\_to\_delay\_insert

Если число неактивных кусков в партиции больше или равно значению `inactive_parts_to_delay_insert`, то `INSERT` искусственно замедляется. Это помогает, когда сервер не может быстро очистить неактивные куски.

Возможные значения:

- Положительное целое число.

Значение по умолчанию: 0 (без ограничений).

## max\_delay\_to\_insert

Величина в секундах, которая используется для расчета задержки `INSERT` в случаях, когда число кусков в партиции больше значения `parts_to_delay_insert`.

Возможные значения:

- Положительное целое число.

Значение по умолчанию: 1.

Величина задержки (в миллисекундах) для `INSERT` вычисляется по формуле:

```
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)
```

Т.е. если в партиции уже 299 кусков и `parts_to_throw_insert` = 300, `parts_to_delay_insert` = 150, а `max_delay_to_insert` = 1, то `INSERT` замедлится на  $\text{pow}(1 * 1000, (1 + 299 - 150) / (300 - 150)) = 1000$  миллисекунд.

## max\_parts\_in\_total

Если суммарное число активных кусков во всех партициях таблицы больше значения `max_parts_in_total`, то `INSERT` прерывается с исключением `Too many parts (N)`.

Возможные значения:

- Положительное целое число.

Значение по умолчанию: 100000.

С большим числом кусков в таблице производительность запросов ClickHouse снижается, а время старта ClickHouse — увеличивается. Чаще всего это следствие неправильного дизайна (ошибки выбора стратегии партиционирования, например, слишком мелкие партиции).

## replicated\_deduplication\_window

Количество хеш-сумм последних вставленных блоков, которые хранятся в Zookeeper.

Возможные значения:

- Положительное целое число.

- 0 (без ограничений).

Значение по умолчанию: 100.

Команда `Insert` создает один или несколько блоков (кусков). При вставке в Replicated таблицы ClickHouse для [дедупликации вставок](#) записывает в Zookeeper хеш-суммы созданных кусков. Но хранятся только последние `replicated_deduplication_window` хеш-сумм. Самые старые хеш-суммы удаляются из Zookeeper.

Большое значение `replicated_deduplication_window` замедляет `Insert`, так как приходится сравнивать большее количество хеш-сумм.

Хеш-сумма рассчитывается по названиям и типам полей, а также по данным вставленного куска (потока байт).

## `non_replicated_deduplication_window`

Количество последних вставленных блоков в нереплицированной [MergeTree](#) таблице, для которых хранятся хеш-суммы для проверки дубликатов.

Возможные значения:

- Положительное целое число.
- 0 (дедупликация отключена).

Значение по умолчанию: 0.

Используется механизм дедупликации, аналогичный реплицированным таблицам (см. описание настройки [replicated\\_deduplication\\_window](#)). Хеш-суммы вставленных кусков записываются в локальный файл на диске.

## `replicated_deduplication_window_seconds`

Время хранения (в секундах) хеш-сумм вставленных блоков в Zookeeper. По истечении этого времени хеш-суммы удаляются.

Возможные значения:

- Положительное целое число.

Значение по умолчанию: 604800 (1 неделя).

Аналогично [replicated\\_deduplication\\_window](#), настройка `replicated_deduplication_window_seconds` задает время хранения хеш-сумм блоков для дедупликации `Insert`. Хеш-суммы старше значения `replicated_deduplication_window_seconds` удаляются из Zookeeper, даже если количество оставшихся хеш-сумм станет меньше чем `replicated_deduplication_window`.

## `old_parts_lifetime`

Время (в секундах) хранения неактивных кусков для защиты от потери данных при спонтанной перезагрузке сервера.

Возможные значения:

- Положительное целое число.

Значение по умолчанию: 480.

После объединения нескольких кусков в один новый ClickHouse помечает исходные куски как неактивные и удаляет их по прошествии `old_parts_lifetime` секунд.

Неактивные куски удаляются, если они не нужны для текущих запросов, т.е. если счетчик ссылок куска `refcount` имеет нулевое значение.

При записи нового куска `fsync` не вызывается, поэтому неактивные куски удаляются позже. Это значит, что некоторое время новый кусок находится только в оперативной памяти сервера (кеш ОС). Если сервер перезагрузится спонтанно, новый слитый кусок может испортиться или потеряться.

Во время запуска сервер ClickHouse проверяет целостность кусков.

Если новый (слитый) кусок поврежден, ClickHouse возвращает неактивные куски в список активных и позже снова выполняет слияние. В этом случае испорченный кусок получает новое имя (добавляется префикс `broken_`) и попадает в каталог `detached`.

Если проверка целостности не выявляет проблем в слитом куске, то исходные неактивные куски переименовываются (добавляется префикс `ignored_`) и перемещаются в каталог `detached`.

Стандартное для Linux значение `dirty_expire_centisecs` — 30 секунд. Это максимальное время, в течение которого записанные данные хранятся только в оперативной памяти. Если нагрузка на дисковую систему большая, то данные записываются намного позже. Значение 480 секунд подобрали экспериментальным путем — это время, за которое новый кусок гарантированно запишется на диск.

## `replicated_fetches_http_connection_timeout`

Тайм-аут HTTP-соединения (в секундах) для запросов на скачивание кусков. Наследуется из профиля по умолчанию `http_connection_timeout`, если не задан явно.

Возможные значения:

- 0 - используется значение `http_connection_timeout`.
- Любое положительное целое число.

Значение по умолчанию: 0.

## `replicated_fetches_http_send_timeout`

Тайм-аут (в секундах) для отправки HTTP-запросов на скачивание кусков. Наследуется из профиля по умолчанию `http_send_timeout`, если не задан явно.

Возможные значения:

- 0 - используется значение `http_send_timeout`.
- Любое положительное целое число.

Значение по умолчанию: 0.

## `replicated_fetches_http_receive_timeout`

Тайм-аут (в секундах) для получения HTTP-запросов на скачивание кусков. Наследуется из профиля по умолчанию `http_receive_timeout`, если не задан явно.

Возможные значения:

- 0 - используется значение `http_receive_timeout`.
- Любое положительное целое число.

Значение по умолчанию: 0.

## `max_replicated_fetches_network_bandwidth`

Ограничивает максимальную скорость скачивания данных в сети (в байтах в секунду) для синхронизаций между [репликами](#). Настройка применяется к конкретной таблице, в отличие от [max\\_replicated\\_fetches\\_network\\_bandwidth\\_for\\_server](#), которая применяется к серверу.

Можно ограничить скорость обмена данными как для всего сервера, так и для конкретной таблицы, но для этого значение табличной настройки должно быть меньше серверной. Иначе сервер будет учитывать только настройку [max\\_replicated\\_fetches\\_network\\_bandwidth\\_for\\_server](#).

Настройка соблюдается неточно.

Возможные значения:

- Любое целое положительное число.
- 0 — Скорость не ограничена.

Значение по умолчанию: 0.

### Использование

Может быть использована для ограничения скорости передачи данных при репликации данных для добавления или замены новых узлов.

## **max\_replicated\_sends\_network\_bandwidth**

Ограничивает максимальную скорость отправки данных по сети (в байтах в секунду) для синхронизации между [репликами](#). Настройка применяется к конкретной таблице, в отличие от [max\\_replicated\\_fetches\\_network\\_bandwidth\\_for\\_server](#), которая применяется к серверу.

Можно ограничить скорость обмена данными как для всего сервера, так и для конкретной таблицы, но для этого значение табличной настройки должно быть меньше серверной. Иначе сервер будет учитывать только настройку [max\\_replicated\\_fetches\\_network\\_bandwidth\\_for\\_server](#).

Настройка следуетя неточно.

Возможные значения:

- Любое целое положительное число.
- 0 — Скорость не ограничена.

Значение по умолчанию: 0.

### Использование

Может быть использована для ограничения скорости сети при репликации данных для добавления или замены новых узлов.

## **max\_bytes\_to\_merge\_at\_max\_space\_in\_pool**

Максимальный суммарный размер кусков (в байтах) в одном слиянии, если есть свободные ресурсы в фоновом пуле.

[max\\_bytes\\_to\\_merge\\_at\\_max\\_space\\_in\\_pool](#) примерно соответствует максимально возможному размеру куска, созданного автоматическим фоновым слиянием.

Возможные значения:

- Положительное целое число.

Значение по умолчанию: 161061273600 (150ГБ).

Планировщик слияний периодически анализирует размер и количество кусков в партициях, и если в пуле хватает ресурсов, то начинает фоновое слияние. Слияния выполняются до тех пор, пока суммарный размер входных кусков не достигнет `max_bytes_to_merge_at_max_space_in_pool`.

Слияния, начатые по [OPTIMIZE FINAL](#), не учитывают `max_bytes_to_merge_at_max_space_in_pool` и объединяют куски пока есть доступные ресурсы (свободное дисковое пространство) до тех пор, пока в партиции не останется один кусок.

## `max_bytes_to_merge_at_min_space_in_pool`

Максимальный суммарный размер кусков (в байтах) в одном слиянии при минимуме свободных ресурсов в фоновом пуле.

Возможные значения:

- Положительное целое число.

Значение по умолчанию: 1048576 (1 МБ).

`max_bytes_to_merge_at_min_space_in_pool` задает максимальный суммарный размер кусков, которые можно объединить несмотря на нехватку свободных ресурсов (дискового пространства) в фоновом пуле. Это нужно, чтобы уменьшить количество маленьких кусков и снизить вероятность ошибки `Too many parts`.

Слияния резервируют дисковое пространство, удваивая суммарный размер кусков в слиянии. Поэтому при малом объеме свободного места на диске может сложиться ситуация, когда свободное место есть, но оно уже зарезервировано текущими слияниями. Из-за этого другие слияния не начинаются, и количество маленьких кусков в партиции растет с каждым запросом `INSERT`.

## `merge_max_block_size`

Количество строк в блоках, которые читаются из объединяемых кусков.

Возможные значения:

- Положительное целое число.

Значение по умолчанию: 8192

Слияние читает строки из кусков блоками по `merge_max_block_size` строк, производит слияние и записывает результат в новый кусок. Читаемый блок помещается в оперативную память, т.е. `merge_max_block_size` влияет на размер оперативной памяти, необходимой для слияния. Таким образом, слияния могут потреблять большое количество оперативной памяти для таблиц, хранящих очень большие строки (если средний размер строки 100кб, то при слиянии 10 кусков будет использовано  $(100\text{кб} * 10 * 8192) = \sim 8\text{ГБ}$  оперативной памяти). Уменьшив `merge_max_block_size`, можно сократить размер оперативной памяти, необходимой для слияния, но при этом процесс слияния замедлится.

## `max_part_loading_threads`

Максимальное количество потоков, которые читают куски при старте ClickHouse.

Возможные значения:

- Положительное целое число.

Значение по умолчанию: определяется автоматически (по количеству ядер процессора).

На старте ClickHouse читает все куски из всех таблиц (читает файлы с метаданными кусков), чтобы построить в оперативной памяти список всех кусков. В некоторых системах с большим количеством кусков этот процесс может занимать длительное время. Это время можно сократить, увеличив `max_part_loading_threads` (если при этом хватает ресурсов процессора и диска).

## `max_partitions_to_read`

Ограничивает максимальное числоパーティций для чтения в одном запросе.

Указанное при создании таблицы значение настройки может быть переназначено настройкой на уровне запроса.

Возможные значения:

- Любое положительное целое число.

Значение по умолчанию: -1 (неограниченно).

## `allow_floating_point_partition_key`

Позволяет использовать число с плавающей запятой в качестве ключа партиционирования.

Возможные значения:

- 0 — Ключ партиционирования с плавающей запятой не разрешен.
- 1 — Ключ партиционирования с плавающей запятой разрешен.

Значение по умолчанию: 0.

## `check_sample_column_is_correct`

Разрешает проверку того, что тип данных столбца для сэмплирования или выражения сэмплирования при создании таблицы верный. Тип данных должен соответствовать одному из беззнаковых [целочисленных типов](#): `UInt8`, `UInt16`, `UInt32`, `UInt64`.

Возможные значения:

- `true` — проверка включена.
- `false` — проверка при создании таблицы не проводится.

Значение по умолчанию: `true`.

По умолчанию сервер ClickHouse при создании таблицы проверяет тип данных столбца для сэмплирования или выражения сэмплирования. Если уже существуют таблицы с некорректным выражением сэмплирования, то чтобы не возникало исключение при запуске сервера, установите `check_sample_column_is_correct` в значение `false`.

# Утилиты ClickHouse

- [clickhouse-local](#) — позволяет выполнять SQL-запросы над данными без остановки сервера ClickHouse, подобно утилите `awk`.
- [clickhouse-copier](#) — копирует (и перешардирует) данные с одного кластера на другой.
- [clickhouse-benchmark](#) — устанавливает соединение с сервером ClickHouse и запускает циклическое выполнение указанных запросов.
- [clickhouse-format](#) — позволяет форматировать входящие запросы.

- [ClickHouse obfuscator](#) — обфусцирует данные.
- [ClickHouse compressor](#) — упаковывает и распаковывает данные.
- [clickhouse-odbc-bridge](#) — прокси-сервер для ODBC.

## clickhouse-copier

Копирует данные из таблиц одногого кластера в таблицы другого (или этого же) кластера.

Можно запустить несколько `clickhouse-copier` для разных серверах для выполнения одного и того же задания. Для синхронизации между процессами используется ZooKeeper.

После запуска, `clickhouse-copier`:

- Соединяется с ZooKeeper и получает:
  - Задания на копирование.
  - Состояние заданий на копирование.
- Выполняет задания.

Каждый запущенный процесс выбирает "ближайший" шард исходного кластера и копирует данные в кластер назначения, при необходимости перешардируя их.

`clickhouse-copier` отслеживает изменения в ZooKeeper и применяет их «на лету».

Для снижения сетевого трафика рекомендуем запускать `clickhouse-copier` на том же сервере, где находятся исходные данные.

## Запуск Clickhouse-copier

Утилиту следует запускать вручную следующим образом:

```
$ clickhouse-copier --daemon --config zookeeper.xml --task-path /task/path --base-dir /path/to/dir
```

Параметры запуска:

- `daemon` - запускает `clickhouse-copier` в режиме демона.
- `config` - путь к файлу `zookeeper.xml` с параметрами соединения с ZooKeeper.
- `task-path` - путь к ноде ZooKeeper. Нода используется для синхронизации между процессами `clickhouse-copier` и для хранения заданий. Задания хранятся в `$task-path/description`.
- `task-file` - необязательный путь к файлу с описанием конфигурации заданий для загрузки в ZooKeeper.
- `task-upload-force` - Загрузить `task-file` в ZooKeeper даже если уже было загружено.
- `base-dir` - путь к логам и вспомогательным файлам. При запуске `clickhouse-copier` создает в `$base-dir` подкаталоги `clickhouse-copier_YYYYMMHHSS_<PID>`. Если параметр не указан, то каталоги будут создаваться в каталоге, где `clickhouse-copier` был запущен.

## Формат 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>

```

## Конфигурация заданий на копирование

```

<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/ru/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 отслеживает изменения /task/path/description и применяет их «на лету». Если вы поменяете, например, значение max\_workers, то количество процессов, выполняющих задания, также изменится.

## clickhouse-local

Принимает на вход данные, которые можно представить в табличном виде и выполняет над ними операции, заданные на [языке запросов ClickHouse](#).

`clickhouse-local` использует движок сервера ClickHouse, т.е. поддерживает все форматы данных и движки таблиц, с которыми работает ClickHouse, при этом для выполнения операций не требуется запущенный сервер.

`clickhouse-local` при настройке по умолчанию не имеет доступа к данным, которыми управляет сервер ClickHouse, установленный на этом же хосте, однако можно подключить конфигурацию сервера с помощью ключа `--config-file`.

## Warning

Мы не рекомендуем подключать серверную конфигурацию к `clickhouse-local`, поскольку данные можно легко повредить неосторожными действиями.

Для временных данных по умолчанию создается специальный каталог.

## Вызов программы

Основной формат вызова:

```
$ clickhouse-local --structure "table_structure" --input-format "format_of_incoming_data" \
--query "query"
```

Ключи команды:

- `-S, --structure` — структура таблицы, в которую будут помещены входящие данные.
- `-if, --input-format` — формат входящих данных. По умолчанию — `TSV`.
- `-f, --file` — путь к файлу с данными. По умолчанию — `stdin`.
- `-q, --query` — запросы на выполнение. Разделитель запросов — `;`.
- `-qf, --queries-file` — путь к файлу с запросами для выполнения. Необходимо задать либо параметр `query`, либо `queries-file`.
- `-N, --table` — имя таблицы, в которую будут помещены входящие данные. По умолчанию — `table`.
- `-of, --format, --output-format` — формат выходных данных. По умолчанию — `TSV`.
- `-d, --database` — база данных по умолчанию. Если не указано, используется значение `_local`.
- `--stacktrace` — вывод отладочной информации при исключениях.
- `--echo` — перед выполнением запрос выводится в консоль.
- `--verbose` — подробный вывод при выполнении запроса.
- `--logger.console` — логирование действий в консоль.
- `--logger.log` — логирование действий в файл с указанным именем.
- `--logger.level` — уровень логирования.
- `--ignore-error` — не прекращать обработку если запрос выдал ошибку.

- `-c, --config-file` — путь к файлу конфигурации. По умолчанию `clickhouse-local` запускается с пустой конфигурацией. Конфигурационный файл имеет тот же формат, что и для сервера ClickHouse, и в нём можно использовать все конфигурационные параметры сервера. Обычно подключение конфигурации не требуется; если требуется установить отдельный параметр, то это можно сделать ключом с именем параметра.
- `--no-system-tables` — запуск без использования системных таблиц.
- `--help` — вывод справочной информации о `clickhouse-local`.
- `-V, --version` — вывод текущей версии и выход.

## Примеры вызова

```
$ 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
```

Вызов выше эквивалентен следующему:

```
$ 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
```

Необязательно использовать ключи `stdin` или `--file`. Вы можете открывать любое количество файлов с помощью **табличной функции** `file`:

```
$ 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
```

Объём оперативной памяти, занимаемой процессами, которые запустил пользователь (Unix):

Запрос:

```
$ 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"
```

Результат:

```
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

Устанавливает соединение с сервером ClickHouse и запускает циклическое выполнение указанных запросов.

### Синтаксис

```
$ clickhouse-benchmark --query ["single query"] [keys]
```

ИЛИ

```
$ echo "single query" | clickhouse-benchmark [keys]
```

ИЛИ

```
$ clickhouse-benchmark [keys] <<< "single query"
```

Если нужно выполнить набор запросов, создайте текстовый файл и расположите каждый запрос на отдельной строке в файле. Например:

```
SELECT * FROM system.numbers LIMIT 10000000;
SELECT 1;
```

После этого передайте этот файл в стандартный ввод `clickhouse-benchmark`:

```
clickhouse-benchmark [keys] < queries_file;
```

## Ключи

- `--query=QUERY` — исполняемый запрос. Если параметр не передан, `clickhouse-benchmark` будет считывать запросы из стандартного ввода.
- `-c N, --concurrency=N` — количество запросов, которые `clickhouse-benchmark` отправляет одновременно. Значение по умолчанию: 1.
- `-d N, --delay=N` — интервал в секундах между промежуточными отчетами (чтобы отключить отчеты, установите 0). Значение по умолчанию: 1.
- `-h HOST, --host=HOST` — хост сервера. Значение по умолчанию: `localhost`. Для **режима сравнения** можно использовать несколько `-h` ключей.
- `-p N, --port=N` — порт сервера. Значение по умолчанию: 9000. Для **режима сравнения** можно использовать несколько `-p` ключей.
- `-i N, --iterations=N` — общее число запросов. Значение по умолчанию: 0 (вечно будет повторяться).

- `-r, --randomize` — использовать случайный порядок выполнения запросов при наличии более одного входного запроса.
- `-s, --secure` — используется TLS соединение.
- `-t N, --timelimit=N` — лимит по времени в секундах. `clickhouse-benchmark` перестает отправлять запросы при достижении лимита по времени. Значение по умолчанию: 0 (лимит отключен).
- `--confidence=N` — уровень доверия для Т-критерия. Возможные значения: 0 (80%), 1 (90%), 2 (95%), 3 (98%), 4 (99%), 5 (99.5%). Значение по умолчанию: 5. В [режиме сравнения](#) `clickhouse-benchmark` проверяет [двуихыборочный t-критерий Стьюдента для независимых выборок](#) чтобы определить, различны ли две выборки при выбранном уровне доверия.
- `--cumulative` — выводить статистику за все время работы, а не за последний временной интервал.
- `--database=DATABASE_NAME` — имя базы данных ClickHouse. Значение по умолчанию: `default`.
- `--json=FILEPATH` — дополнительный вывод в формате JSON. Когда этот ключ указан, `clickhouse-benchmark` выводит отчет в указанный JSON-файл.
- `--user=USERNAME` — имя пользователя ClickHouse. Значение по умолчанию: `default`.
- `--password=PSWD` — пароль пользователя ClickHouse. Значение по умолчанию: пустая строка.
- `--stacktrace` — вывод трассировки стека исключений. Когда этот ключ указан, `clickhouse-benchmark` выводит трассировку стека исключений.
- `--stage=WORD` — стадия обработки запроса на сервере. ClickHouse останавливает обработку запроса и возвращает ответ `clickhouse-benchmark` на заданной стадии. Возможные значения: `complete, fetch_columns, with_mergeable_state`. Значение по умолчанию: `complete`.
- `--help` — показывает справку.

Если нужно применить [настройки](#) для запросов, их можно передать как ключ `--<session setting name>=SETTING_VALUE`. Например, `--max_memory_usage=1048576`.

## Вывод

По умолчанию, `clickhouse-benchmark` выводит сообщение для каждого `--delay` интервала.

Пример сообщения:

```
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.
```

В сообщении можно найти:

- Количество запросов в поле `Queries executed`:
- Стока статуса, содержащая (в таком же порядке):
  - Endpoint сервера ClickHouse.
  - Число обработанных запросов.
  - QPS: количество запросов, выполняемых сервером за секунду в течение `--delay` интервала.
  - RPS: количество строк, читаемых сервером за секунду в течение `--delay` интервала.
  - MiB/s: количество Мебибайтов, считываемых сервером за секунду в течение `--delay` интервала.
  - result RPS: количество строк, добавленное сервером в результат запроса за секунду в течение `--delay` интервала.
  - result MiB/s: количество Мебибайтов, размещаемое сервером в результат запроса за секунду в течение `--delay` интервала.
- Процентили времени выполнения запросов.

## Режим сравнения

`clickhouse-benchmark` может сравнивать производительность двух работающих серверов ClickHouse.

Для использования сравнительного режима укажите конечную точку двух серверов двумя парами ключей `--host`, `--port`. Связь ключей соответствует позициям в списке аргументов: первый `--host` соответствует первому `--port` и так далее. `clickhouse-benchmark` устанавливает соединение с обоими серверами и отсылает запросы. Каждый запрос адресован случайно выбранному серверу. Результаты выводятся отдельно для каждого сервера.

## Пример

```
$ 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

Позволяет форматировать входящие запросы.

Ключи:

- `--help` или `-h` — выводит описание ключей.
- `--hilite` — добавляет подсветку синтаксиса с экранированием символов.
- `--oneline` — форматирование в одну строку.
- `--quiet` или `-q` — проверяет синтаксис без вывода результата.
- `--multiquery` or `-n` — поддерживает несколько запросов в одной строке.
- `--obfuscate` — обfuscates вместо форматирования.
- `--seed <строка>` — задает строку, которая определяет результат обфускации.
- `--backslash` — добавляет обратный слеш в конце каждой строки отформатированного запроса.  
Удобно использовать если многострочный запрос скопирован из интернета или другого источника и его нужно выполнить из командной строки.

# Примеры

- Подсветка синтаксиса и форматирование в одну строку:

```
$ clickhouse-format --oneline --hilite <<< "SELECT sum(number) FROM numbers(5);"
```

Результат:

```
SELECT sum(number) FROM numbers(5)
```

- Несколько запросов в одной строке:

```
$ clickhouse-format -n <<< "SELECT * FROM (SELECT 1 AS x UNION ALL SELECT 1 UNION DISTINCT SELECT 3);"
```

Результат:

```
SELECT *
FROM
(
    SELECT 1 AS x
    UNION ALL
    SELECT 1
    UNION DISTINCT
    SELECT 3
)
;
```

- Обfuscация:

```
$ clickhouse-format --seed Hello --obfuscate <<< "SELECT cost_first_screen BETWEEN a AND b, CASE WHEN x >= 123 THEN y ELSE NULL END;"
```

Результат:

```
SELECT treasury_mammoth_hazelnut BETWEEN nutmeg AND span, CASE WHEN chive >= 116 THEN switching ELSE ANYTHING END;
```

Тот же запрос с другой инициализацией обфускатора:

```
$ clickhouse-format --seed World --obfuscate <<< "SELECT cost_first_screen BETWEEN a AND b, CASE WHEN x >= 123 THEN y ELSE NULL END;"
```

Результат:

```
SELECT horse_tape_summer BETWEEN folklore AND moccasins, CASE WHEN intestine >= 116 THEN nonconformist ELSE FORESTRY END;
```

- Добавление обратного слеша:

```
$ clickhouse-format --backslash <<< "SELECT * FROM (SELECT 1 AS x UNION ALL SELECT 1 UNION DISTINCT SELECT 3);"
```

Результат:

```
SELECT * \
FROM \
( \
    SELECT 1 AS x \
    UNION ALL \
    SELECT 1 \
    UNION DISTINCT \
    SELECT 3 \
)
```

## ClickHouse compressor

Simple program for data compression and decompression in ClickHouse way.

### 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-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 # Result in RowBinary format  
12246623837185725195925621517
```

## Обфускатор ClickHouse

Простой инструмент для обфускации табличных данных.

Он считывает данные входной таблицы и создает выходную таблицу, которая сохраняет некоторые свойства входных данных, но при этом содержит другие данные.

Это позволяет публиковать практически реальные данные и использовать их в тестах на производительность.

Обфускатор предназначен для сохранения следующих свойств данных:

- кардинальность (количество уникальных данных) для каждого столбца и каждого кортежа столбцов;
- условная кардинальность: количество уникальных данных одного столбца в соответствии со значением другого столбца;
- вероятностные распределения абсолютного значения целых чисел; знак числа типа Int; показатель степени и знак для чисел с плавающей запятой;
- вероятностное распределение длины строк;
- вероятность нулевых значений чисел; пустые строки и массивы, `NULL`;
- степень сжатия данных алгоритмом LZ77 и семейством энтропийных кодеков;
- непрерывность (величина разницы) значений времени в таблице; непрерывность значений с плавающей запятой;
- дату из значений `DateTime`;
- кодировка UTF-8 значений строки;
- строковые значения выглядят естественным образом.

Большинство перечисленных выше свойств пригодны для тестирования производительности. Чтение данных, фильтрация, агрегирование и сортировка будут работать почти с той же скоростью, что и исходные данные, благодаря сохраненной кардинальности, величине, степени сжатия и т. д.

Он работает детерминированно. Вы задаёте значение инициализатора, а преобразование полностью определяется входными данными и инициализатором.

Некоторые преобразования выполняются один к одному, и их можно отменить. Поэтому нужно использовать большое значение инициализатора и хранить его в секрете.

Обфускатор использует некоторые криптографические примитивы для преобразования данных, но, с криптографической точки зрения, результат будет небезопасным. В нем могут сохраниться данные, которые не следует публиковать.

Он всегда оставляет без изменений числа 0, 1, -1, даты, длины массивов и нулевые флаги. Например, если у вас есть столбец `IsMobile` в таблице со значениями 0 и 1, то в преобразованных данных он будет иметь то же значение.

Таким образом, пользователь сможет посчитать точное соотношение мобильного трафика.

Давайте рассмотрим случай, когда у вас есть какие-то личные данные в таблице (например, электронная почта пользователя), и вы не хотите их публиковать.

Если ваша таблица достаточно большая и содержит несколько разных электронных почтовых адресов, и ни один из них не встречается часто, то обфускатор полностью анонимизирует все данные. Но, если у вас есть небольшое количество разных значений в столбце, он может скопировать некоторые из них.

В этом случае вам следует посмотреть на алгоритм работы инструмента и настроить параметры командной строки.

Обфускатор полезен в работе со средним объемом данных (не менее 1000 строк).

## [экспериментально] Поддержка OpenTelemetry

ClickHouse поддерживает [OpenTelemetry](#) — открытый стандарт для сбора трассировок и метрик из распределенного приложения.

### Предупреждение

Поддержка стандарта экспериментальная и будет со временем меняться.

## Обеспечение поддержки контекста трассировки в ClickHouse

ClickHouse принимает контекстную информацию трассировки через HTTP заголовок `tracecontext`, как описано в [рекомендации W3C](#). Также он принимает контекстную информацию через нативный протокол, который используется для связи между серверами ClickHouse или между клиентом и сервером. Для ручного тестирования стандартный заголовок `tracecontext`, содержащий контекст трассировки, может быть передан в `clickhouse-client` через флаги: `--opentelemetry-traceparent` и `--opentelemetry-tracestate`.

Если входящий контекст трассировки не указан, ClickHouse может начать трассировку с вероятностью, задаваемой настройкой [`opentelemetry\_start\_trace\_probability`](#).

## Распространение контекста трассировки

Контекст трассировки распространяется на нижестоящие сервисы в следующих случаях:

- При использовании запросов к удаленным серверам ClickHouse, например, при использовании движка таблиц [Distributed](#).
- При использовании табличной функции [url](#). Информация контекста трассировки передается в HTTP заголовки.

## Как ClickHouse выполняет трассировку

ClickHouse создает `trace spans` для каждого запроса и некоторых этапов выполнения запроса, таких как планирование запросов или распределенные запросы.

Чтобы анализировать информацию трассировки, ее следует экспортить в систему мониторинга, поддерживающую OpenTelemetry, такую как [Jaeger](#) или [Prometheus](#). ClickHouse не зависит от конкретной системы мониторинга, вместо этого предоставляя данные трассировки только через системную таблицу. Информация о диапазоне трассировки в OpenTelemetry, [требуемая стандартом](#), хранится в системной таблице `system.opentelemetry_span_log`.

Таблица должна быть включена в конфигурации сервера, смотрите элемент `opentelemetry_span_log` в файле конфигурации `config.xml`. По умолчанию таблица включена всегда.

Теги или атрибуты сохраняются в виде двух параллельных массивов, содержащих ключи и значения. Для работы с ними используйте [ARRAY JOIN](#).

## Типы кеша

При выполнении запросов ClickHouse использует различные типы кеша.

Основные типы кеша:

- `mark_cache` — кеш засечек, используемых движками таблиц семейства [MergeTree](#).
- `uncompressed_cache` — кеш несжатых данных, используемых движками таблиц семейства [MergeTree](#).

Дополнительные типы кеша:

- DNS-кеш.
- Кеш данных формата [regexp](#).
- Кеш скомпилированных выражений.
- Кеш схем формата [Avro](#).
- Кеш данных в [словарях](#).

Непрямое использование:

- Кеш страницы ОС.

Чтобы очистить кеш, используйте выражение [SYSTEM DROP ... CACHE](#).

## [пре-продакшн] ClickHouse Keeper

Сервер ClickHouse использует сервис координации [ZooKeeper](#) для [репликации](#) данных и выполнения [распределенных DDL запросов](#). ClickHouse Keeper — это альтернативный сервис координации, совместимый с ZooKeeper.

### Предупреждение

ClickHouse Keeper находится в стадии пре-продакшн и тестируется в CI ClickHouse и на нескольких внутренних инсталляциях.

## Детали реализации

ZooKeeper — один из первых широко известных сервисов координации с открытым исходным кодом. Он реализован на языке программирования Java, имеет достаточно простую и мощную модель данных. Алгоритм координации Zookeeper называется ZAB (ZooKeeper Atomic Broadcast). Он не гарантирует линеаризуемость операций чтения, поскольку каждый узел ZooKeeper обслуживает чтения локально. В отличие от ZooKeeper, ClickHouse Keeper реализован на C++ и использует алгоритм **RAFT**, **реализация**. Этот алгоритм позволяет достичь линеаризуемости чтения и записи, имеет несколько реализаций с открытым исходным кодом на разных языках.

По умолчанию ClickHouse Keeper предоставляет те же гарантии, что и ZooKeeper (линеаризуемость записей, последовательная согласованность чтений). У него есть совместимый клиент-серверный протокол, поэтому любой стандартный клиент ZooKeeper может использоваться для взаимодействия с ClickHouse Keeper. Снэпшоты и журналы имеют несовместимый с ZooKeeper формат, однако можно конвертировать данные Zookeeper в снэпшот ClickHouse Keeper с помощью `clickhouse-keeper-converter`. Межсерверный протокол ClickHouse Keeper также несовместим с ZooKeeper, поэтому создание смешанного кластера ZooKeeper / ClickHouse Keeper невозможно.

## Конфигурация

ClickHouse Keeper может использоваться как равноценная замена ZooKeeper или как внутренняя часть сервера ClickHouse, но в обоих случаях конфигурация представлена файлом .xml. Главный тег конфигурации ClickHouse Keeper — это `<keeper_server>`. Параметры конфигурации:

- `tcp_port` — порт для подключения клиента (по умолчанию для ZooKeeper: 2181).
- `tcp_port_secure` — зашифрованный порт для подключения клиента.
- `server_id` — уникальный идентификатор сервера, каждый участник кластера должен иметь уникальный номер (1, 2, 3 и т. д.).
- `log_storage_path` — путь к журналам координации, лучше хранить их на незанятоом устройстве (актуально и для ZooKeeper).
- `snapshot_storage_path` — путь к снэпшотам координации.

Другие общие параметры наследуются из конфигурации сервера ClickHouse (`listen_host`, `logger`, и т. д.).

Настройки внутренней координации находятся в `<keeper_server>.<coordination_settings>`:

- `operation_timeout_ms` — максимальное время ожидания для одной клиентской операции в миллисекундах (по умолчанию: 10000).
- `session_timeout_ms` — максимальное время ожидания для клиентской сессии в миллисекундах (по умолчанию: 30000).
- `dead_session_check_period_ms` — частота, с которой ClickHouse Keeper проверяет мертвые сессии и удаляет их, в миллисекундах (по умолчанию: 500).
- `heart_beat_interval_ms` — частота, с которой узел-лидер ClickHouse Keeper отправляет хартбиты узлам-последователям, в миллисекундах (по умолчанию: 500).
- `election_timeout_lower_bound_ms` — время, после которого последователь может инициировать выборы лидера, если не получил от него сердцебиения (по умолчанию: 1000).
- `election_timeout_upper_bound_ms` — время, после которого последователь должен инициировать выборы лидера, если не получил от него сердцебиения (по умолчанию: 2000).
- `rotate_log_storage_interval` — количество записей в журнале координации для хранения в одном файле (по умолчанию: 100000).

- `reserved_log_items` — минимальное количество записей в журнале координации которые нужно сохранять после снятия снепшота (по умолчанию: 100000).
- `snapshot_distance` — частота, с которой ClickHouse Keeper делает новые снэпшоты (по количеству записей в журналах), в миллисекундах (по умолчанию: 100000).
- `snapshots_to_keep` — количество снэпшотов для сохранения (по умолчанию: 3).
- `stale_log_gap` — время, после которого лидер считает последователя устаревшим и отправляет ему снэпшот вместо журналов (по умолчанию: 10000).
- `fresh_log_gap` — максимальное отставание от лидера в количестве записей журнала после которого последователь считает себя не отстающим (по умолчанию: 200).
- `max_requests_batch_size` — количество запросов на запись, которые будут сгруппированы в один перед отправкой через RAFT (по умолчанию: 100).
- `force_sync` — вызывать `fsync` при каждой записи в журнал координации (по умолчанию: `true`).
- `quorum_reads` — выполнять запросы чтения аналогично запросам записи через весь консенсус RAFT с негативным эффектом на производительность и размер журналов (по умолчанию: `false`).
- `raft_logs_level` — уровень логгирования сообщений в текстовый лог (`trace`, `debug` и т. д.) (по умолчанию: `information`).
- `auto_forwarding` — разрешить пересылку запросов на запись от последователей лидеру (по умолчанию: `true`).
- `shutdown_timeout` — время ожидания завершения внутренних подключений и выключения, в миллисекундах (по умолчанию: 5000).
- `startup_timeout` — время отключения сервера, если он не подключается к другим участникам кворума, в миллисекундах (по умолчанию: 30000).

Конфигурация кворума находится в `<keeper_server>.RAFT_CONFIGURATION` и содержит описание серверов. Единственный параметр для всего кворума — `secure`, который включает зашифрованное соединение для связи между участниками кворума. Параметры для каждого `<server>`:

- `id` — идентификатор сервера в кворуме.
- `hostname` — имя хоста, на котором размещен сервер.
- `port` — порт, на котором серверу доступны соединения для внутренней коммуникации.

Примеры конфигурации кворума с тремя узлами можно найти в [интеграционных тестах](#) с префиксом `test_keeper_`. Пример конфигурации для сервера №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>
```

## Как запустить

ClickHouse Keeper входит в пакет `clickhouse-server`, просто добавьте конфигурацию `<keeper_server>` и запустите сервер ClickHouse как обычно. Если вы хотите запустить ClickHouse Keeper автономно, сделайте это аналогичным способом:

```
clickhouse-keeper --config /etc/your_path_to_config/config.xml --daemon
```

## [экспериментально] Переход с ZooKeeper

Плавный переход с ZooKeeper на ClickHouse Keeper невозможен, необходимо остановить кластер ZooKeeper, преобразовать данные и запустить ClickHouse Keeper. Утилита `clickhouse-keeper-converter` конвертирует журналы и снэпшоты ZooKeeper в снэпшот ClickHouse Keeper. Работа утилиты проверена только для версий ZooKeeper выше 3.4. Для миграции необходимо выполнить следующие шаги:

1. Остановите все узлы ZooKeeper.
2. Необязательно, но рекомендуется: найдите узел-лидер ZooKeeper, запустите и снова остановите его. Это заставит ZooKeeper создать консистентный снэпшот.
3. Запустите `clickhouse-keeper-converter` на лидере, например:

```
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. Скопируйте снэпшот на узлы сервера ClickHouse с настроенным `keeper` или запустите ClickHouse Keeper вместо ZooKeeper. Снэпшот должен сохраняться на всех узлах: в противном случае пустые узлы могут захватить лидерство и сконвертированные данные могут быть отброшены на старте.

# Хранение данных на внешних дисках

Данные, которые обрабатываются в ClickHouse, обычно хранятся в файловой системе локально, где развернут сервер ClickHouse. При этом для хранения данных требуются диски большого объема, которые могут быть довольно дорогостоящими. Решением проблемы может стать хранение данных отдельно от сервера — в распределенных файловых системах — [Amazon S3](#) или Hadoop ([HDFS](#)).

Для работы с данными, хранящимися в файловой системе Amazon S3, используйте движок [S3](#), а для работы с данными в файловой системе Hadoop — движок [HDFS](#).

## Репликация без копирования данных

Для дисков [S3](#) и [HDFS](#) в ClickHouse поддерживается репликация без копирования данных (zero-copy): если данные хранятся на нескольких репликах, то при синхронизации пересылаются только метаданные (пути к кускам данных), а сами данные не копируются.

## Использование сервиса HDFS для хранения данных

Таблицы семейств [MergeTree](#) и [Log](#) могут хранить данные в сервисе HDFS при использовании диска типа [HDFS](#).

Пример конфигурации:

```
<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>
```

Обязательные параметры:

- `endpoint` — URL точки приема запроса на стороне HDFS в формате `path`. URL точки должен содержать путь к корневой директории на сервере, где хранятся данные.

Необязательные параметры:

- `min_bytes_for_seek` — минимальное количество байтов, которые используются для операций поиска вместо последовательного чтения. Значение по умолчанию: 1 Мбайт.

## Использование виртуальной файловой системы для шифрования данных

Вы можете зашифровать данные, сохраненные на внешних дисках **S3** или **HDFS** или на локальном диске. Чтобы включить режим шифрования, в конфигурационном файле вы должны указать диск с типом `encrypted` и тип диска, на котором будут сохранены данные. Диск типа `encrypted` шифрует данные "на лету", то есть при чтении файлов с этого диска расшифровка происходит автоматически. Таким образом, вы можете работать с диском типа `encrypted` как с обычным.

Пример конфигурации:

```
<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>
```

Например, когда ClickHouse записывает данные из какой-либо таблицы в файл `store/all_1_1_0/data.bin` на `disk1`, то на самом деле этот файл будет записан на физический диск по пути `/path1/store/all_1_1_0/data.bin`.

При записи того же файла на диск `disk2` он будет записан на физический диск в зашифрованном виде по пути `/path1/path2/store/all_1_1_0/data.bin`.

Обязательные параметры:

- `type` — `encrypted`. Иначе зашифрованный диск создан не будет.
- `disk` — тип диска для хранения данных.
- `key` — ключ для шифрования и расшифровки. Тип: `UInt64`. Вы можете использовать параметр `key_hex` для шифрования в шестнадцатеричной форме.  
Вы можете указать несколько ключей, используя атрибут `id` (смотрите пример выше).

Необязательные параметры:

- `path` — путь к месту на диске, где будут сохранены данные. Если не указан, данные будут сохранены в корневом каталоге.
- `current_key_id` — ключ, используемый для шифрования. Все указанные ключи могут быть использованы для расшифровки, и вы всегда можете переключиться на другой ключ, сохраняя доступ к ранее зашифрованным данным.
- `algorithm` — алгоритм шифрования данных. Возможные значения: `AES_128_CTR`, `AES_192_CTR` или `AES_256_CTR`. Значение по умолчанию: `AES_128_CTR`. Длина ключа зависит от алгоритма: `AES_128_CTR` — 16 байт, `AES_192_CTR` — 24 байта, `AES_256_CTR` — 32 байта.

Пример конфигурации:

```

<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">00112233445566778899aabccddeff</key_hex>
        <key_hex id="1">ffeeddccbbaa99887766554433221100</key_hex>
        <current_key_id>1</current_key_id>
      </disk_s3_encrypted>
    </disks>
  </storage_configuration>
</clickhouse>

```

## Хранение данных на веб-сервере

Существует утилита `clickhouse-static-files-uploader`, которая готовит каталог данных для данной таблицы (`SELECT data_paths FROM system.tables WHERE name = 'table_name'`). Для каждой таблицы, необходимой вам, вы получаете каталог файлов. Эти файлы могут быть загружены, например, на веб-сервер в виде статических файлов. После этой подготовки вы можете загрузить эту таблицу на любой сервер ClickHouse через `DiskWeb`.

Этот тип диска используется только для чтения, то есть данные на нем неизменяемы. Новая таблица загружается с помощью запроса `ATTACH TABLE` (см. пример ниже) и при каждом чтении данные будут доставаться по заданному `url` через `http` запрос, то есть локально данные не хранятся. Любое изменение данных приведет к исключению, т.е. не допускаются следующие типы запросов: `CREATE TABLE`, `ALTER TABLE`, `RENAME TABLE`, `DETACH TABLE` и `TRUNCATE TABLE`.

Хранение данных на веб-сервере поддерживается только для табличных движков семейства `MergeTree` и `Log`. Чтобы получить доступ к данным, хранящимся на диске `web`, при выполнении запроса используйте настройку `storage_policy`. Например, `ATTACH TABLE table_web UUID '{}' (id Int32) ENGINE = MergeTree() ORDER BY id SETTINGS storage_policy = 'web'.`

Готовый тестовый пример. Добавьте эту конфигурацию в 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>

```

А затем выполните этот запрос:

```

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,
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 storage_policy='web';

```

Обязательные параметры:

- **type** — `web`. Иначе диск создан не будет.
- **endpoint** — URL точки приема запроса в формате `path`. URL точки должен содержать путь к корневой директории на сервере для хранения данных, куда эти данные были загружены.

Необязательные параметры:

- `min_bytes_for_seek` — минимальное количество байтов, которое используются для операций поиска вместо последовательного чтения. Значение по умолчанию: 1 Mb.
- `remote_fs_read_backoff_threshold` — максимальное время ожидания при попытке чтения данных с удаленного диска. Значение по умолчанию: 10000 секунд.
- `remote_fs_read_backoff_max_tries` — максимальное количество попыток чтения данных с задержкой. Значение по умолчанию: 5.

Если после выполнения запроса генерируется исключение DB:Exception Unreachable URL, то могут помочь настройки: `http_connection_timeout`, `http_receive_timeout`, `keep_alive_timeout`.

Чтобы получить файлы для загрузки, выполните:

```
clickhouse static-files-disk-uploader --metadata-path <path> --output-dir <dir> (--metadata-path может быть  
получен в результате запроса SELECT data_paths FROM system.tables WHERE name = 'table_name').
```

Файлы должны быть загружены по пути `<endpoint>/store/`, но конфигурация должна содержать только `endpoint`.

Если URL-адрес недоступен при загрузке на диск, когда сервер запускает таблицы, то все ошибки будут пойманы. Если в этом случае были ошибки, то таблицы можно перезагрузить (сделать видимыми) с помощью `DETACH TABLE table_name -> ATTACH TABLE table_name`. Если метаданные были успешно загружены при запуске сервера, то таблицы будут доступны сразу.

Чтобы ограничить количество попыток чтения данных во время одного HTTP-запроса, используйте настройку `http_max_single_read_retries`.

---

`toc_priority: 58`  
`toc_title: Usage Recommendations`

## ClickHouse Development

### Инструкция для разработчиков

Сборка ClickHouse поддерживается на Linux, FreeBSD, Mac OS X.

#### Если вы используете Windows

Если вы используете Windows, вам потребуется создать виртуальную машину с Ubuntu. Для работы с виртуальной машиной, установите VirtualBox. Скачать Ubuntu можно на сайте: <https://www.ubuntu.com/#download> Создайте виртуальную машину из полученного образа. Выделите для неё не менее 4 GB оперативной памяти. Для запуска терминала в Ubuntu, найдите в меню программу со словом terminal (gnome-terminal, konsole или что-то в этом роде) или нажмите Ctrl+Alt+T.

#### Если вы используете 32-битную систему

ClickHouse не работает и не собирается на 32-битных системах. Получите доступ к 64-битной системе и продолжайте.

#### Создание репозитория на GitHub

Для работы с репозиторием ClickHouse, вам потребуется аккаунт на GitHub. Наверное, он у вас уже есть.

Если аккаунта нет - зарегистрируйтесь на <https://github.com/>. Создайте ssh ключи, если их нет, и загрузите публичные ключи на GitHub. Это потребуется для отправки изменений. Для работы с GitHub можно использовать такие же ssh ключи, как и для работы с другими ssh серверами - скорее всего, они уже у вас есть.

Создайте fork репозитория ClickHouse. Для этого, на странице <https://github.com/ClickHouse/ClickHouse> нажмите на кнопку «fork» в правом верхнем углу. Вы получите полную копию репозитория ClickHouse на своём аккаунте, которая называется «форк». Процесс разработки состоит в том, чтобы внести нужные изменения в свой форк репозитория, а затем создать «pull request» для принятия изменений в основной репозиторий.

Для работы с git репозиториями, установите git.

В Ubuntu выполните в терминале:

```
sudo apt update  
sudo apt install git
```

Краткое руководство по использованию Git: <https://education.github.com/git-cheat-sheet-education.pdf>

Подробное руководство по использованию Git: <https://git-scm.com/book/ru/v2>

## Клонирование репозитория на рабочую машину

Затем вам потребуется загрузить исходники для работы на свой компьютер. Это называется «клонирование репозитория», потому что создаёт на вашем компьютере локальную копию репозитория, с которой вы будете работать.

Выполните в терминале:

```
git clone git@github.com:ClickHouse/ClickHouse.git  
cd ClickHouse
```

Замените первое вхождение слова ClickHouse в команде для git на имя вашего аккаунта на GitHub.

Эта команда создаст директорию ClickHouse, содержащую рабочую копию проекта.

Необходимо, чтобы путь к рабочей копии не содержал пробелы в именах директорий. Это может привести к проблемам в работе системы сборки.

Обратите внимание, что репозиторий ClickHouse использует submodules. Так называются ссылки на дополнительные репозитории (например, внешние библиотеки, от которых зависит проект). Это значит, что при клонировании репозитория, следует указывать ключ --recursive, как в примере выше. Если репозиторий был клонирован без submodules, то для их скачивания, необходимо выполнить:

```
git submodule init  
git submodule update
```

Проверить наличие submodules можно с помощью команды `git submodule status`.

Если вы получили сообщение об ошибке:

```
Permission denied (publickey).  
fatal: Could not read from remote repository.  
  
Please make sure you have the correct access rights  
and the repository exists.
```

Как правило это означает, что отсутствуют ssh ключи для соединения с GitHub. Ключи расположены в директории `~/.ssh`. В интерфейсе GitHub, в настройках, необходимо загрузить публичные ключи, чтобы он их понимал.

Вы также можете клонировать репозиторий по протоколу https:

```
git clone https://github.com/ClickHouse/ClickHouse.git
```

Этот вариант не подходит для отправки изменений на сервер. Вы можете временно его использовать, а затем добавить ssh ключи и заменить адрес репозитория с помощью команды `git remote`.

Вы можете также добавить для своего локального репозитория адрес оригинального репозитория Яндекса, чтобы притягивать оттуда обновления:

```
git remote add upstream git@github.com:ClickHouse/ClickHouse.git
```

После этого, вы сможете добавлять в свой репозиторий обновления из репозитория Яндекса с помощью команды `git pull upstream master`.

## Работа с сабмодулями Git

Работа с сабмодулями git может быть достаточно болезненной. Следующие команды позволят содержать их в порядке:

```
# ! Каждая команда принимает аргумент
# Обновить URLs удалённого репозитория для каждого сабмодуля, используется относительно редко
git submodule sync
# Добавить новые сабмодули
git submodule init
# Обновить сабмодули до актуального состояния
git submodule update
# Две последние команды могут быть объединены вместе:
git submodule update --init
```

Следующие команды помогут сбросить все сабмодули в изначальное состояние (!ВНИМАНИЕ! - все изменения в сабмодулях будут утеряны):

```
# Synchronizes submodules' remote URL with .gitmodules
# Обновить URLs удалённого репозитория для каждого сабмодуля
git submodule sync
# Обновить существующие модули и добавить отсутствующие
git submodule update --init
# Удалить все изменения в сабмодуле относительно HEAD
git submodule foreach git reset --hard
# Очистить игнорируемые файлы
git submodule foreach git clean -xfd
# Повторить последние 4 команды для каждого из сабмодулей
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
```

## Система сборки

ClickHouse использует систему сборки CMake и Ninja.

CMake - генератор задач сборки.

Ninja - система запуска сборочных задач.

Для установки на Ubuntu или Debian, Mint, выполните `sudo apt install cmake ninja-build`.

Для установки на CentOS, RedHat, выполните `sudo yum install cmake ninja-build`.

Если у вас Arch или Gentoo, то вы сами знаете, как установить CMake.

Для установки CMake и Ninja на Mac OS X, сначала установите Homebrew, а затем, с помощью него, установите всё остальное.

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"  
brew install cmake ninja
```

Проверьте версию CMake: `cmake --version`. Если версия меньше 3.12, то установите новую версию с сайта <https://cmake.org/download/>

## Необязательные внешние библиотеки

ClickHouse использует для сборки некоторое количество внешних библиотек. Но ни одну из них не требуется отдельно устанавливать, так как они собираются вместе с ClickHouse, из исходников, которые расположены в `submodules`. Посмотреть набор этих библиотек можно в директории `contrib`.

## Компилятор C++

В качестве компилятора C++ поддерживается Clang начиная с версии 11.

Впрочем, наша среда continuous integration проверяет около десятка вариантов сборки, включая gcc, но сборка с помощью gcc непригодна для использования в продакшене.

На Ubuntu и Debian вы можете использовать скрипт для автоматической установки (см. [официальный сайт](#))

```
sudo bash -c "$(wget -O - https://apt.llvm.org/llvm.sh)"
```

Сборка под Mac OS X поддерживается только для компилятора Clang. Чтобы установить его выполните `brew install llvm`

## Процесс сборки

Теперь вы готовы к сборке ClickHouse. Для размещения собранных файлов, рекомендуется создать отдельную директорию `build` внутри директории ClickHouse:

```
mkdir build  
cd build
```

Вы можете иметь несколько разных директорий (`build_release`, `build_debug`) для разных вариантов сборки.

Находясь в директории `build`, выполните конфигурацию сборки с помощью CMake.

Перед первым запуском необходимо выставить переменные окружения, отвечающие за выбор компилятора.

```
export CC=clang CXX=clang++  
cmake ..
```

Переменная CC отвечает за компилятор С (сокращение от слов C Compiler), переменная CXX отвечает за выбор компилятора С++ (символ Х - это как плюс, но положенный набок, ради того, чтобы превратить его в букву). При получении ошибки типа Could not find compiler set in environment variable CC: clang необходимо указать в значениях для переменных CC и CXX явную версию компилятора, например, clang-12 и clang++-12.

Для более быстрой сборки, можно использовать debug вариант - сборку без оптимизаций. Для этого, укажите параметр -D CMAKE\_BUILD\_TYPE=Debug:

```
cmake -D CMAKE_BUILD_TYPE=Debug ..
```

В случае использования на разработческой машине старого HDD или SSD, а также при желании использовать меньше места для артефактов сборки можно использовать следующую команду:

```
cmake -DUSE_DEBUG_HELPERS=1 -DUSE_STATIC_LIBRARIES=0 -DSPLIT_SHARED_LIBRARIES=1 -  
DCLICKHOUSE_SPLIT_BINARY=1 ..
```

При этом надо учесть, что получаемые в результате сборки исполняемые файлы будут динамически слинкованы с библиотеками, и поэтому фактически станут непереносимыми на другие компьютеры (либо для этого нужно будет предпринять значительно больше усилий по сравнению со статической сборкой). Плюсом же в данном случае является значительно меньшее время сборки (это проявляется не на первой сборке, а на последующих, после внесения изменений в исходный код - тратится меньшее время на линковку по сравнению со статической сборкой) и значительно меньшее использование места на жёстком диске (экономия более, чем в 3 раза по сравнению со статической сборкой). Для целей разработки, когда планируются только отладочные запуски на том же компьютере, где осуществлялась сборка, это может быть наиболее удобным вариантом.

Вы можете изменить вариант сборки, выполнив новую команду в директории build.

Запустите ninja для сборки:

```
ninja clickhouse-server clickhouse-client
```

В этом примере собираются только нужные в первую очередь программы.

Если вы хотите собрать все программы (утилиты и тесты), то запустите ninja без параметров:

```
ninja
```

Для полной сборки требуется около 30 GB свободного места на диске или 15 GB для сборки только основных программ.

При наличии небольшого количества оперативной памяти на компьютере, следует ограничить количество параллельных задач с помощью параметра -j:

```
ninja -j 1 clickhouse-server clickhouse-client
```

На машинах с 4 GB памяти, рекомендуется указывать значение 1, а если памяти до 8 GB, укажите значение 2.

Если вы получили сообщение `ninja: error: loading 'build.ninja': No such file or directory`, значит конфигурация сборки прошла с ошибкой и вам необходимо посмотреть на сообщение об ошибке выше.

В случае успешного запуска, вы увидите прогресс сборки - количество обработанных задач и общее количество задач.

В процессе сборки могут появится сообщения `libprotobuf WARNING` про protobuf файлы в библиотеке `libhdfs2`. Это не имеет значения.

В случае получения ошибок вида `error: variable 'y' set but not used [-Werror,-Wunused-but-set-variable]` нужно попробовать использовать другую версию компилятора clang. Например, на момент написания данного текста описанная выше команда по установке clang для Ubuntu 20.04 по-умолчанию устанавливает clang-13, с которым возникает эта ошибка. Для решения проблемы можно установить clang-12 с помощью команд:

```
wget https://apt.llvm.org/llvm.sh  
chmod +x llvm.sh  
sudo ./llvm.sh 12
```

И далее использовать именно его, указав соответствующую версию при установке переменных окружения CC и CXX перед вызовом стаке.

При успешной сборке, вы получите готовый исполняемый файл `ClickHouse/build/programs/clickhouse`:

```
ls -l programs/clickhouse
```

## Запуск собранной версии ClickHouse

Для запуска сервера из под текущего пользователя, с выводом логов в терминал и с использованием примеров конфигурационных файлов, расположенных в исходниках, перейдите в директорию `ClickHouse/programs/server/` (эта директория находится не в директории build) и выполните:

```
../../build/programs/clickhouse server
```

В этом случае, ClickHouse будет использовать конфигурационные файлы, расположенные в текущей директории. Вы можете запустить `clickhouse server` из любой директории, передав ему путь к конфигурационному файлу в аргументе командной строки `--config-file`.

Для подключения к ClickHouse с помощью `clickhouse-client`, в соседнем терминале, зайдите в директорию `ClickHouse/build/programs/` и выполните `./clickhouse client`.

Если вы получили сообщение `Connection refused` на Mac OS X или FreeBSD, то укажите для клиента `127.0.0.1` в качестве имени хоста:

```
clickhouse client --host 127.0.0.1
```

Вы можете заменить собранным вами ClickHouse продакшен версию, установленную в системе. Для этого, установите ClickHouse на свою машину по инструкции с официального сайта. Затем выполните:

```
sudo service clickhouse-server stop  
sudo cp ClickHouse/build/programs/clickhouse /usr/bin/  
sudo service clickhouse-server start
```

Обратите внимание, что `clickhouse-client`, `clickhouse-server` и другие, являются симлинками на общий бинарник `clickhouse`.

Также вы можете запустить собранный вами ClickHouse с конфигурационным файлом системного ClickHouse:

```
sudo service clickhouse-server stop  
sudo -u clickhouse ClickHouse/build/programs/clickhouse server --config-file /etc/clickhouse-server/config.xml
```

## Среда разработки

Если вы не знаете, какую среду разработки использовать, то рекомендуется использовать CLion. CLion является платным ПО, но его можно использовать бесплатно в течение пробного периода. Также он бесплатен для учащихся. CLion можно использовать как под Linux, так и под Mac OS X.

Также в качестве среды разработки, вы можете использовать KDevelop или QT Creator. KDevelop - очень удобная, но нестабильная среда разработки. Если KDevelop вылетает через небольшое время после открытия проекта, вам следует нажать на кнопку «Stop All» как только он открыл список файлов проекта. После этого, KDevelop можно будет использовать.

В качестве простых редакторов кода можно использовать Sublime Text или Visual Studio Code или Kate (все варианты доступны под Linux).

На всякий случай заметим, что CLion самостоятельно создаёт свою build директорию, самостоятельно выбирает тип сборки debug по-умолчанию, для конфигурации использует встроенную в CLion версию CMake вместо установленного вами, а для запуска задач использует make вместо ninja (но при желании начиная с версии CLion 2019.3 EAP можно настроить использование ninja, см. подробнее [тут](#)). Это нормально, просто имейте это ввиду, чтобы не возникало путаницы.

## Написание кода

Описание архитектуры ClickHouse: <https://clickhouse.com/docs/ru/development/architecture/>

Стиль кода: <https://clickhouse.com/docs/ru/development/style/>

Рекомендации по добавлению сторонних библиотек и поддержанию в них пользовательских изменений: <https://clickhouse.com/docs/ru/development/contrib/#adding-third-party-libraries>

Разработка тестов: <https://clickhouse.com/docs/ru/development/tests/>

Список задач: <https://github.com/ClickHouse/ClickHouse/issues?q=is%3Aopen+is%3Aissue+label%3A%22easy+task%22>

## Тестовые данные

Разработка ClickHouse часто требует загрузки реалистичных наборов данных. Особенно это важно для тестирования производительности. Специально для вас мы подготовили набор данных, представляющий собой анонимизированные данные Яндекс.Метрики. Загрузка этих данных потребует ещё 3 GB места на диске. Для выполнения большинства задач разработки, загружать эти данные не обязательно.

```
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 TABLE test.hits ( WatchID UInt64, JavaEnable UInt8, Title String, GoodEvent Int16, EventTime DateTime,  
EventDate Date, CounterID UInt32, ClientID UInt32, ClientIP FixedString(16), RegionID UInt32, UserIP UInt64
```

```

EventDate Date, CounterID UInt32, ClientIP UInt32, ClientIPFixedAsString(10), 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, AdvEngineld 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.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, AdvEngineld UInt8, PlaceID
Int32, RefererCategories Array(UInt16), URLCategories Array(UInt16), URLRegions Array(UInt32), RefererRegions
Array(UInt32), IsYandex UInt8, GoalReachesDepth Int32, GoalReachesURL Int32, GoalReachesAny Int32,
SocialSourceNetworkID UInt8, SocialSourcePage String, MobilePhoneModel String, ClientEventTime DateTime,
RegionID UInt32, ClientIP UInt32, ClientIP6 FixedString(16), RemoteIP UInt32, RemoteIP6 FixedString(16),
IPNetworkID UInt32, SilverlightVersion3 UInt32, CodeVersion UInt32, ResolutionWidth UInt16, ResolutionHeight
UInt16, UserAgentMajor UInt16, UserAgentMinor UInt16, WindowClientWidth UInt16, WindowClientHeight UInt16,
SilverlightVersion2 UInt8, SilverlightVersion4 UInt16, FlashVersion3 UInt16, FlashVersion4 UInt16, ClientTimeZone
Int16, OS UInt8, UserAgent UInt8, ResolutionDepth UInt8, FlashMajor UInt8, FlashMinor UInt8, NetMajor UInt8,
NetMinor UInt8, MobilePhone UInt8, SilverlightVersion1 UInt8, Age UInt8, Sex UInt8, Income UInt8, JavaEnable
UInt8, CookieEnable UInt8, JavascriptEnable UInt8, IsMobile UInt8, BrowserLanguage UInt16, BrowserCountry
UInt16, Interests UInt16, Robotness UInt8, GeneralInterests Array(UInt16), Params Array(String), `Goals.ID`
Array(UInt32), `Goals.Serial` Array(UInt32), `Goals.EventTime` Array(DateTime), `Goals.Price` Array(Int64),
`Goals.OrderID` Array(String), `Goals.CurrencyID` Array(UInt32), WatchIDs Array(UInt64), ParamSumPrice Int64,
ParamCurrency FixedString(3), ParamCurrencyID UInt16, ClickLogID UInt64, ClickEventID Int32, ClickGoodEvent
Int32, ClickEventTime DateTime, ClickPriorityID Int32, ClickPhraseID Int32, ClickPageID Int32, ClickPlaceID Int32,
ClickTypeID Int32, ClickResourceID Int32, ClickCost UInt32, ClickClientIP UInt32, ClickDomainID UInt32, ClickURL
String, ClickAttempt UInt8, ClickOrderID UInt32, ClickBannerID UInt32, ClickMarketCategoryID UInt32,
ClickMarketPP UInt32, ClickMarketCategoryName String, ClickMarketPPName String, ClickAWAPSCampaignName
String, ClickPageName String, ClickTargetType UInt16, ClickTargetPhraseID UInt64, ClickContextType UInt8,
ClickSelectType Int8, ClickOptions String, ClickGroupBannerID Int32, OpenstatServiceName String,
OpenstatCampaignID String, OpenstatAdID String, OpenstatSourceID String, UTMSource String, UTMMedium String,
UTMCampaign String, UTMContent String, UTMTerm String, FromTag String, HasGCLID UInt8, FirstVisit DateTime,
PredLastVisit Date, LastVisit Date, TotalVisits UInt32, `TraficSource.ID` Array(Int8), `TraficSource.SearchEngineID`
Array(UInt16), `TraficSource.AdvEngineld` 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

```

# Создание Pull Request

Откройте свой форк репозитория в интерфейсе GitHub. Если вы вели разработку в бранче, выберите этот бранч. На странице будет доступна кнопка «Pull request». По сути, это означает «создать заявку на принятие моих изменений в основной репозиторий».

Pull request можно создать, даже если работа над задачей ещё не завершена. В этом случае, добавьте в его название слово «WIP» (work in progress). Название можно будет изменить позже. Это полезно для совместного просмотра и обсуждения изменений, а также для запуска всех имеющихся тестов. Введите краткое описание изменений - впоследствии, оно будет использовано для релизных changelog.

Тесты будут запущены, как только сотрудники Яндекса поставят для pull request тег «Can be tested». Результаты первых проверок (стиль кода) появятся уже через несколько минут. Результаты сборки появятся примерно через пол часа. Результаты основного набора тестов будут доступны в пределах часа.

Система подготовит сборки ClickHouse специально для вашего pull request. Для их получения, нажмите на ссылку «Details» у проверки «Clickhouse build check». Там вы сможете найти прямые ссылки на собранные .deb пакеты ClickHouse, которые, при желании, вы даже сможете установить на свои продакшен серверы (если не страшно).

Вероятнее всего, часть сборок не будет успешной с первого раза. Ведь мы проверяем сборку кода и gcc и clang, а при сборке с помощью clang включаются почти все существующие в природе warnings (всегда с флагом -Werror). На той же странице, вы сможете найти логи сборки - вам не обязательно самому собирать ClickHouse всеми возможными способами.

## 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.

## Обзор архитектуры ClickHouse

ClickHouse - полноценная колоночная СУБД. Данные хранятся в колонках, а в процессе обработки - в массивах (векторах или фрагментах (chunk'ах) колонок). По возможности операции выполняются на массивах, а не на индивидуальных значениях. Это называется "векторизованное выполнения запросов" (vectorized query execution), и помогает снизить стоимость фактической обработки данных.

Эта идея не нова. Такой подход использовался в [APL](#) (A programming language, 1957) и его потомках: [A +](#) (диалект APL), [J](#) (1990), [K](#) (1993) и [Q](#) (язык программирования Kx Systems, 2003).

Программирование на массивах (Array programming) используется в научных вычислительных системах. Эта идея не является чем-то новым и для реляционных баз данных: например, она используется в системе [VectorWise](#) (так же известной как Actian Vector Analytic Database от Actian Corporation).

Существует два различных подхода для увеличения скорости обработки запросов: выполнение векторизованного запроса и генерация кода во время выполнения (runtime code generation). В последнем случае код генерируется на лету для каждого типа запроса, удаляя все косвенные и динамические обращения. Ни один из этих подходов не имеет явного преимущества. Генерация кода во время выполнения выигрывает, если объединяет большое число операций, таким образом полностью используя вычислительные блоки и конвейер CPU. Выполнение векторизованного запроса может быть менее практично потому, что действует временные векторы данных, которые должны быть записаны и прочитаны из кэша. Если временные данные не помещаются в L2 кэш, будут проблемы. С другой стороны выполнение векторизованного запроса упрощает использование SIMD инструкций CPU. [Научная работа](#) наших друзей показывает преимущества сочетания обоих подходов. ClickHouse использует выполнение векторизованного запроса и имеет ограниченную начальную поддержку генерации кода во время выполнения.

## Колонки

Для представления столбцов в памяти (фактически, фрагментов столбцов) используется интерфейс `IColumn`. Интерфейс предоставляет вспомогательные методы для реализации различных реляционных операторов. Почти все операции иммутабельные: они не изменяют оригинальных колонок, а создают новую с измененными значениями. Например, метод `IColumn :: filter` принимает фильтр - набор байт. Он используется для реляционных операторов `WHERE` и `HAVING`. Другой пример: метод `IColumn :: permute` используется для поддержки `ORDER BY`, метод `IColumn :: cut` - `LIMIT` и т. д.

Различные реализации `IColumn` (`ColumnUInt8`, `ColumnString` и т. д.) отвечают за распределение данных колонки в памяти. Для колонок целочисленного типа это один смежный массив, такой как `std :: vector`. Для колонок типа `String` и `Array` это два вектора: один для всех элементов массивов, располагающихся смежно, второй для хранения смещения до начала каждого массива. Также существует реализация `ColumnConst`, в которой хранится только одно значение в памяти, но выглядит как колонка.

## Поля

Тем не менее, можно работать и с индивидуальными значениями. Для представления индивидуальных значений используется Поле (`Field`). `Field` - размеченное объединение `UInt64`, `Int64`, `Float64`, `String` и `Array`. `IColumn` имеет метод оператор `[]` для получения значения по индексу `n` как `Field`, а также метод `insert` для добавления `Field` в конец колонки. Эти методы не очень эффективны, так как требуют временных объектов `Field`, представляющих индивидуальное значение. Есть более эффективные методы, такие как `insertFrom`, `insertRangeFrom` и т.д.

`Field` не несет в себе достаточно информации о конкретном типе данных в таблице. Например, `UInt8`, `UInt16`, `UInt32` и `UInt64` в `Field` представлены как `UInt64`.

## Дырявые абстракции (Leaky Abstractions)

`IColumn` предоставляет методы для общих реляционных преобразований данных, но они не отвечают всем потребностям. Например, `ColumnUInt64` не имеет метода для вычисления суммы двух столбцов, а `ColumnString` не имеет метода для запуска поиска по подстроке. Эти бесчисленные процедуры реализованы вне `IColumn`.

Различные функции на колонках могут быть реализованы обобщенным, неэффективным путем, используя `IColumn` методы для извлечения значений `Field`, или специальным путем, используя знания о внутреннем распределении данных в памяти в конкретной реализации `IColumn`. Для этого функции приводятся к конкретному типу `IColumn` и работают напрямую с его внутренним представлением. Например, в `ColumnUInt64` есть метод `getData`, который возвращает ссылку на внутренний массив, чтение и заполнение которого, выполняется отдельной процедурой напрямую. Фактически, мы имеем "дырявые абстракции", обеспечивающие эффективные специализации различных процедур.

# Типы данных (Data Types)

`IDataType` отвечает за сериализацию и десериализацию - чтение и запись фрагментов колонок или индивидуальных значений в бинарном или текстовом формате.

`IDataType` точно соответствует типам данных в таблицах - `DataTypeUInt32`, `DataTypeDateTime`, `DataTypeString` и т. д.

`IDataType` и `IColumn` слабо связаны друг с другом. Различные типы данных могут быть представлены в памяти с помощью одной реализации `IColumn`. Например, и `DataTypeUInt32`, и `DataTypeDateTime` в памяти представлены как `ColumnUInt32` или `ColumnConstUInt32`. В добавок к этому, один тип данных может быть представлен различными реализациями `IColumn`. Например, `DataTypeUInt8` может быть представлен как `ColumnUInt8` и `ColumnConstUInt8`.

`IDataType` хранит только метаданные. Например, `DataTypeUInt8` не хранит ничего (кроме скрытого указателя `vptr`), а `DataTypeFixedString` хранит только `N` (фиксированный размер строки).

В `IDataType` есть вспомогательные методы для данных различного формата. Среди них методы сериализации значений, допускающих использование кавычек, сериализации значения в JSON или XML. Среди них нет прямого соответствия форматам данных. Например, различные форматы `Pretty` и `TabSeparated` могут использовать один вспомогательный метод `serializeTextEscaped` интерфейса `IDataType`.

## Блоки (Block)

`Block` это контейнер, который представляет фрагмент (chunk) таблицы в памяти. Это набор троек - (`IColumn`, `IDataType`, имя колонки). В процессе выполнения запроса, данные обрабатываются `Block`-ами. Если у нас есть `Block`, значит у нас есть данные (в объекте `IColumn`), информация о типе (в `IDataType`), которая говорит нам, как работать с колонкой, и имя колонки (оригинальное имя колонки таблицы или служебное имя, присвоенное для получения промежуточных результатов вычислений).

При вычислении некоторой функции на колонках в блоке мы добавляем еще одну колонку с результатами в блок, не трогая колонки аргументов функции, потому что операции иммутабельные. Позже ненужные колонки могут быть удалены из блока, но не модифицированы. Это удобно для устранения общих подвыражений.

Блоки создаются для всех обработанных фрагментов данных. Напоминаем, что одни и те же типы вычислений, имена колонок и типы переиспользуются в разных блоках и только данные колонок изменяются. Лучше разделить данные и заголовок блока потому, что в блоках маленького размера мы имеем большой оверхэд по временным строкам при копировании умных указателей (`shared_ptrs`) и имен колонок.

## Потоки блоков (Block Streams)

Потоки блоков обрабатывают данные. Мы используем потоки блоков для чтения данных, трансформации или записи данных куда-либо. `IBlockInputStream` предоставляет метод `read` для получения следующего блока, пока это возможно, и метод `write`, чтобы продвигать (push) блок куда-либо.

Потоки отвечают за:

1. Чтение и запись в таблицу. Таблица лишь возвращает поток для чтения или записи блоков.
2. Реализацию форматов данных. Например, при выводе данных в терминал в формате `Pretty`, вы создаете выходной поток блоков, который форматирует поступающие в него блоки.

3. Трансформацию данных. Допустим, у вас есть `IBlockInputStream` и вы хотите создать отфильтрованный поток. Вы создаете `FilterBlockInputStream` и инициализируете его вашим потоком. Затем вы тянете (pull) блоки из `FilterBlockInputStream`, а он тянет блоки исходного потока, фильтрует их и возвращает отфильтрованные блоки вам. Таким образом построены конвейеры выполнения запросов.

Имеются и более сложные трансформации. Например, когда вы тянете блоки из `AggregatingBlockInputStream`, он считывает все данные из своего источника, агрегирует их, и возвращает поток агрегированных данных вам. Другой пример: конструктор `UnionBlockInputStream` принимает множество источников входных данных и число потоков. Такой Stream работает в несколько потоков и читает данные источников параллельно.

Потоки блоков используют «втягивающий» (pull) подход к управлению потоком выполнения: когда вы вытягиваете блок из первого потока, он, следовательно, вытягивает необходимые блоки из вложенных потоков, так и работает весь конвейер выполнения. Ни «pull» ни «push» не имеют явного преимущества, потому что поток управления неявный, и это ограничивает в реализации различных функций, таких как одновременное выполнение нескольких запросов (слияние нескольких конвейеров вместе). Это ограничение можно преодолеть с помощью сопрограмм (coroutines) или просто запуском дополнительных потоков, которые ждут друг друга. У нас может быть больше возможностей, если мы сделаем поток управления явным: если мы локализуем логику для передачи данных из одной расчетной единицы в другую вне этих расчетных единиц. Читайте эту [статью](#) для углубленного изучения.

Следует отметить, что конвейер выполнения запроса создает временные данные на каждом шаге. Мы стараемся сохранить размер блока достаточно маленьким, чтобы временные данные помещались в кэш процессора. При таком допущении запись и чтение временных данных практически бесплатны по сравнению с другими расчетами. Мы могли бы рассмотреть альтернативу, которая заключается в том, чтобы объединить многие операции в конвейере вместе. Это может сделать конвейер как можно короче и удалить большую часть временных данных, что может быть преимуществом, но у такого подхода также есть недостатки. Например, разделенный конвейер позволяет легко реализовать кэширование промежуточных данных, использование промежуточных данных из аналогичных запросов, выполняемых одновременно, и объединение конвейеров для аналогичных запросов.

## Форматы

Форматы данных реализуются с помощью потоков блоков. Есть форматы представления (presentational), пригодные только для вывода данных клиенту, такие как `Pretty` формат, который предоставляет только `IBlockOutputStream`. И есть форматы ввода/вывода, такие как `TabSeparated` или `JSONEachRow`.

Существуют также потоки строк: `IRowInputStream` и `IRowOutputStream`. Они позволяют вытягивать/выталкивать данные отдельными строками, а не блоками. Они нужны только для упрощения реализации ориентированных на строки форматов. Обертка `BlockInputStreamFromRowInputStream` и `BlockOutputStreamFromRowOutputStream` позволяет конвертировать потоки, ориентированные на строки, в обычные потоки, ориентированные на блоки.

## I/O

Для байт-ориентированных ввода/вывода существуют абстрактные классы `ReadBuffer` и `WriteBuffer`. Они используются вместо C++ `iostream`. Не волнуйтесь: каждый зрелый проект C++ использует что-то другое вместо `iostream` по уважительным причинам.

`ReadBuffer` и `WriteBuffer` это просто непрерывный буфер и курсор, указывающий на позицию в этом буфере. Реализации могут как владеть так и не владеть памятью буфера. Существует виртуальный метод заполнения буфера следующими данными (для `ReadBuffer`) или сброса буфера куда-нибудь (например `WriteBuffer`). Виртуальные методы редко вызываются.

Реализации `ReadBuffer`/`WriteBuffer` используются для работы с файлами и файловыми дескрипторами, а также сетевыми сокетами, для реализации сжатия (`CompressedWriteBuffer` инициализируется вместе с другим `WriteBuffer` и осуществляет сжатие данных перед записью в него), и для других целей – названия `ConcatReadBuffer`, `LimitReadBuffer`, и `HashingWriteBuffer` говорят сами за себя.

Буфера чтения/записи имеют дело только с байтами. В заголовочных файлах `ReadHelpers` и `WriteHelpers` объявлены некоторые функции, чтобы помочь с форматированием ввода/вывода. Например, есть помощники для записи числа в десятичном формате.

Давайте посмотрим, что происходит, когда вы хотите вывести результат в JSON формате в стандартный вывод (`stdout`). У вас есть результирующий набор данных, готовый к извлечению из `IBlockInputStream`. Вы создаете `WriteBufferFromDescriptor(STDOUT_FILENO)` чтобы записать байты в `stdout`. Вы создаете `JSONRowOutputStream`, инициализируете с этим `WriteBuffer`'ом, чтобы записать строки JSON в `stdout`. Кроме того вы создаете `BlockOutputStreamFromRowOutputStream`, реализуя `IBlockOutputStream`. Затем вызывается `copyData` для передачи данных из `IBlockInputStream` в `IBlockOutputStream` и все работает. Внутренний `JSONRowOutputStream` будет писать в формате JSON различные разделители и вызвать `IDataType::serializeTextJSON` метод со ссылкой на `IColumn` и номер строки в качестве аргументов. Следовательно, `IDataType::serializeTextJSON` вызовет метод из `WriteHelpers.h`: например, `writeText` для числовых типов и `writeJSONString` для `Data.TypeString`.

## Таблицы

Интерфейс `IStorage` служит для отображения таблицы. Различные движки таблиц являются реализациями этого интерфейса. Примеры `StorageMergeTree`, `StorageMemory` и так далее. Экземпляры этих классов являются просто таблицами.

Ключевые методы `IStorage` это `read` и `write`. Есть и другие варианты - `alter`, `rename`, `drop` и так далее. Метод `read` принимает следующие аргументы: набор столбцов для чтения из таблицы, `AST` запрос и желаемое количество потоков для вывода. Он возвращает один или несколько объектов `IBlockInputStream` и информацию о стадии обработки данных, которая была завершена внутри табличного движка во время выполнения запроса.

В большинстве случаев метод `read` отвечает только за чтение указанных столбцов из таблицы, а не за дальнейшую обработку данных. Вся дальнейшая обработка данных осуществляется интерпретатором запросов и не входит в сферу ответственности `IStorage`.

Но есть и заметные исключения:

- `AST` запрос, передающийся в метод `read`, может использоваться движком таблицы для получения информации о возможности использования индекса и считывания меньшего количества данных из таблицы.
- Иногда движок таблиц может сам обрабатывать данные до определенного этапа. Например, `StorageDistributed` можно отправить запрос на удаленные серверы, попросить их обработать данные до этапа, когда данные с разных удаленных серверов могут быть объединены, и вернуть эти предварительно обработанные данные. Затем интерпретатор запросов завершает обработку данных.

Метод `read` может возвращать несколько объектов `IBlockInputStream`, позволяя осуществлять параллельную обработку данных. Эти несколько блочных входных потоков могут считываться из таблицы параллельно. Затем вы можете обернуть эти потоки различными преобразованиями (такими как вычисление выражений или фильтрация), которые могут быть вычислены независимо, и создать `UnionBlockInputStream` поверх них, чтобы читать из нескольких потоков параллельно.

Есть и другие варианты. Например, `TableFunction` возвращает временный объект `IStorage`, который можно подставить во `FROM`.

Чтобы получить быстрое представление о том, как реализовать свой движок таблиц, посмотрите на что-то простое, например `StorageMemory` или `StorageTinyLog`.

В качестве результата выполнения метода `read`, `IStorage` возвращает `QueryProcessingStage` – информацию о том, какие части запроса были обработаны внутри хранилища.

## Парсеры (Parsers)

Написанный от руки парсер, анализирующий запрос, работает по методу рекурсивного спуска. Например, `ParserSelectQuery` просто рекурсивно вызывает нижестоящие парсеры для различных частей запроса. Парсеры создают абстрактное синтаксическое дерево (AST). AST представлен узлами, которые являются экземплярами `IAST`.

Генераторы парсеров не используются по историческим причинам.

## Интерпретаторы

Интерпретаторы отвечают за создание конвейера выполнения запроса из `AST`. Есть простые интерпретаторы, такие как `InterpreterExistsQuery` и `InterpreterDropQuery` или более сложный `InterpreterSelectQuery`. Конвейер выполнения запроса представляет собой комбинацию входных и выходных потоков блоков. Например, результатом интерпретации `SELECT` запроса является `IBlockInputStream` для чтения результирующего набора данных; результат интерпретации `INSERT` запроса - это `IBlockOutputStream`, для записи данных, предназначенных для вставки; результат интерпретации `INSERT SELECT` запроса - это `IBlockInputStream`, который возвращает пустой результирующий набор при первом чтении, но копирует данные из `SELECT` к `INSERT`.

`InterpreterSelectQuery` использует `ExpressionAnalyzer` и `ExpressionActions` механизмы для анализа запросов и преобразований. Именно здесь выполняется большинство оптимизаций запросов на основе правил. `ExpressionAnalyzer` написан довольно грязно и должен быть переписан: различные преобразования запросов и оптимизации должны быть извлечены в отдельные классы, чтобы позволить модульные преобразования или запросы.

## ФУНКЦИИ

Существуют обычные функции и агрегатные функции. Агрегатные функции смотрите в следующем разделе.

Обычные функции не изменяют число строк и работают так, как если бы обрабатывали каждую строку независимо. В действительности же, функции вызываются не к отдельным строкам, а блокам данных для реализации векторизованного выполнения запросов.

Некоторые функции, такие как `blockSize`, `rowNumberInBlock`, и `runningAccumulate`, эксплуатируют блочную обработку и нарушают независимость строк.

ClickHouse имеет сильную типизацию, поэтому нет никакого неявного преобразования типов. Если функция не поддерживает определенную комбинацию типов, она создает исключение. Но функции могут работать (перегружаться) для многих различных комбинаций типов. Например, функция `plus` (для реализации + оператор) работает для любой комбинации числовых типов: `UInt8 + Float32`, `UInt16 + Int8` и так далее. Кроме того, некоторые вариадические функции, такие как `concat`, могут принимать любое количество аргументов.

Реализация функции может быть немного неудобной, поскольку функция явно определяет поддерживаемые типы данных и поддерживается `IColumns`. Например, в `plus` функция имеет код, генерируемый экземпляром шаблона C++ для каждой комбинации числовых типов, а также постоянные или непостоянные левые и правые аргументы.

Это отличное место для реализации генерации кода во время выполнения, чтобы избежать раздувания кода шаблона. Кроме того, он позволяет добавлять слитые функции, такие как `fused multiply-add` или выполнять несколько сравнений в одной итерации цикла.

Из-за векторизованного выполнения запроса функции не закорачиваются. Например, если вы пишете `WHERE f(x) AND g(y)`, обе части вычисляются, даже для строк, когда `f(x)` равно нулю (за исключением тех случаев, когда `f(x)` является нулевым постоянным выражением). Но если избирательность условия `f(x)` высока, и расчет `f(x)` обходится гораздо дешевле, чем `g(y)`, лучше всего разделить вычисление на этапы. На первом этапе вычислить `f(x)`, отфильтровать результирующие столбцы, а затем вычислять `g(y)` только для меньших, отфильтрованных фрагментов данных.

## Агрегатные функции

Агрегатные функции - это функции с состоянием (`stateful`). Они накапливают переданные значения в некотором состоянии и позволяют получать результаты из этого состояния. Работа с ними осуществляется с помощью интерфейса `IAggregateFunction`. Состояния могут быть как простыми (состояние для `AggregateFunctionCount` это всего лишь одна переменная типа `UInt64`) так и довольно сложными (состояние `AggregateFunctionUniqCombined` представляет собой комбинацию линейного массива, хэш-таблицы и вероятностной структуры данных `HyperLogLog`).

Состояния распределяются в `Arena` (пул памяти) для работы с несколькими состояниями при выполнении запроса `GROUP BY` высокой кардинальности (большим числом уникальных данных). Состояния могут иметь нетривиальный конструктор и деструктор: например, сложные агрегатные состояния могут сами аллоцировать дополнительную память. Поэтому к созданию и уничтожению состояний, правильной передаче владения и порядку уничтожения следует уделять больше внимания.

Агрегатные состояния могут быть сериализованы и десериализованы для передачи их по сети во время выполнения распределенного запроса или для записи их на диск при дефиците оперативной памяти. Они даже могут храниться в таблице с `DataTypeAggregateFunction`, чтобы позволяют выполнять инкрементное агрегирование данных.

Формат сериализации данных для состояний агрегатных функций в настоящее время не версионируется. Это нормально, если агрегатные состояния хранятся только временно. Но у нас есть такая возможность `AggregatingMergeTree` механизм таблиц для инкрементной агрегации, и люди уже используют его в эксплуатации. Именно по этой причине требуется помнить об обратной совместимости при изменении формата сериализации для любой агрегатной функции.

## Сервер

Сервер предоставляет несколько различных интерфейсов.

- HTTP интерфейс для любых сторонних клиентов.

- TCP интерфейс для родного ClickHouse клиента и межсерверной взаимодействия при выполнении распределенных запросов.
- Интерфейс для передачи данных при репликации.

Внутри простой многопоточный сервер без корутин (coroutines), файберов (fibers) и т.д. Поскольку сервер не предназначен для обработки большого количества простых запросов, а ориентирован на обработку сложных запросов относительно низкой интенсивности, каждый из потоков может обрабатывать огромное количество аналитических запросов.

Сервер инициализирует класс `Context`, где хранит необходимое для выполнения запроса окружение: список доступных баз данных, пользователей и прав доступа, настройки, кластеры, список процессов, журнал запросов и т.д. Это окружение используется интерпретаторами.

Мы поддерживаем полную обратную и прямую совместимость для TCP интерфейса: старые клиенты могут общаться с новыми серверами, а новые клиенты могут общаться со старыми серверами. Но мы не хотим поддерживать его вечно и прекращаем поддержку старых версий примерно через год.

## Note

Для всех сторонних приложений мы рекомендуем использовать HTTP интерфейс, потому что он прост и удобен в использовании. TCP интерфейс тесно связан с внутренними структурами данных: он использует внутренний формат для передачи блоков данных и использует специальное кадрирование для сжатых данных. Мы не выпустили библиотеку С для этого протокола, потому что потребовала бы линковки большей части кодовой базы ClickHouse, что непрактично.

## Выполнение распределенных запросов (Distributed Query Execution)

Сервера в кластере в основном независимы. Вы можете создать Распределенную (Distributed) таблицу на одном или всех серверах в кластере. Такая таблица сама по себе не хранит данные - она только предоставляет возможность "просмотра" всех локальных таблиц на нескольких узлах кластера. При выполнении `SELECT` распределенная таблица переписывает запрос, выбирает удаленные узлы в соответствии с настройками балансировки нагрузки и отправляет им запрос. Распределенная таблица просит удаленные сервера обработать запрос до той стадии, когда промежуточные результаты с разных серверов могут быть объединены. Затем он получает промежуточные результаты и объединяет их. Распределенная таблица пытается возложить как можно больше работы на удаленные серверы и сократить объем промежуточных данных, передаваемых по сети.

Ситуация усложняется при использовании подзапросов в случае `IN` или `JOIN`, когда каждый из них использует таблицу `Distributed`. Есть разные стратегии для выполнения таких запросов.

Глобального плана выполнения распределенных запросов не существует. Каждый узел имеет собственный локальный план для своей части работы. У нас есть простое одностороннее выполнение распределенных запросов: мы отправляем запросы на удаленные узлы и затем объединяем результаты. Но это невозможно для сложных запросов `GROUP BY` высокой кардинальности или запросов с большим числом временных данных в `JOIN`: в таких случаях нам необходимо перераспределить («reshuffle») данные между серверами, что требует дополнительной координации. ClickHouse не поддерживает выполнение запросов такого рода, и нам нужно работать над этим.

## Merge Tree

MergeTree - это семейство движков хранения, поддерживающих индексацию по первичному ключу. Первичный ключ может быть произвольным набором (кортежем) столбцов или выражений. Данные в таблице MergeTree хранятся "частями" ("parts"). Каждая часть хранит данные отсортированные по первичному ключу (данные упорядочены лексикографически). Все столбцы таблицы хранятся в отдельных файлах `column.bin` в этих частях. Файлы состоят из сжатых блоков. Каждый блок обычно содержит от 64 КБ до 1 МБ несжатых данных, в зависимости от среднего значения размера данных. Блоки состоят из значений столбцов, расположенных последовательно один за другим. Значения столбцов находятся в одинаковом порядке для каждого столбца (порядок определяется первичным ключом), поэтому, когда вы выполняете итерацию по многим столбцам, вы получаете значения для соответствующих строк.

Сам первичный ключ является "разреженным" ("sparse"). Он не относится к каждой отдельной строке, а только к некоторым диапазонам данных. Отдельный файл «`primary.idx`» имеет значение первичного ключа для каждой N-й строки, где N называется гранулярностью индекса ("index\_granularity", обычно N = 8192). Также для каждого столбца у нас есть файлы `column.mrk` с "метками" ("marks"), которые обозначают смещение для каждой N-й строки в файле данных. Каждая метка представляет собой пару: смещение начала сжатого блока от начала файла и смещение к началу данных в распакованном блоке. Обычно сжатые блоки выравниваются по меткам, а смещение в распакованном блоке равно нулю. Данные для `primary.idx` всегда находятся в памяти, а данные для файлов `column.mrk` кэшируются.

Когда мы собираемся читать что-то из части данных MergeTree, мы смотрим содержимое `primary.idx` и определяем диапазоны, которые могут содержать запрошенные данные, затем просматриваем содержимое `column.mrk` и вычисляем смещение, чтобы начать чтение этих диапазонов. Из-за разреженности могут быть прочитаны лишние данные. ClickHouse не подходит для простых точечных запросов высокой интенсивности, потому что весь диапазон строк размером `index_granularity` должен быть прочитан для каждого ключа, а сжатый блок должен быть полностью распакован для каждого столбца. Мы сделали индекс разреженным, потому что мы должны иметь возможность поддерживать триллионы строк на один сервер без существенных расходов памяти на индексацию. Кроме того, поскольку первичный ключ является разреженным, он не уникален: он не может проверить наличие ключа в таблице во время `INSERT`. Вы можете иметь множество строк с одним и тем же ключом в таблице.

При выполнении `INSERT` для группы данных в MergeTree, элементы группы сортируются по первичному ключу и образуют новую "часть". Фоновые потоки периодически выбирают некоторые части и объединяют их в одну отсортированную часть, чтобы сохранить относительно небольшое количество частей. Вот почему он называется MergeTree. Конечно, объединение приводит к повышению интенсивности записи. Все части иммутабельные: они только создаются и удаляются, но не изменяются. Когда выполняется `SELECT`, он содержит снимок таблицы (набор частей). После объединения старые части также сохраняются в течение некоторого времени, чтобы упростить восстановление после сбоя, поэтому, если мы видим, что какая-то объединенная часть, вероятно, повреждена, мы можем заменить ее исходными частями.

MergeTree не является деревом LSM (Log-structured merge-tree — журнально-структурированное дерево со слиянием), потому что оно не содержит «`memtable`» и «`log`»: вставленные данные записываются непосредственно в файловую систему. Это делает его пригодным только для вставки данных в пакетах, а не по отдельным строкам и не очень часто - примерно раз в секунду это нормально, а тысячу раз в секунду - нет. Мы сделали это для простоты и потому, что мы уже вставляем данные в пакеты в наших приложениях.

Таблицы MergeTree могут иметь только один (первичный) индекс: вторичных индексов нет. Было бы неплохо разрешить несколько физических представлениям в одной логической таблице, например, хранить данные в более чем одном физическом порядке или даже разрешить представления с предварительно агрегированными данными вместе с исходными данными.

Существуют движки MergeTree, которые выполняют дополнительную работу во время фоновых слияний. Примерами являются CollapsingMergeTree и AggregatingMergeTree. Это можно рассматривать как специальную поддержку обновления. Помните, что это не настоящие обновления, поскольку пользователи обычно не контролируют время выполнения фоновых слияний, а данные в таблице MergeTree почти всегда хранятся в нескольких частях, а не в полностью объединенной форме.

## Репликация

Репликация в ClickHouse может быть настроена для каждой таблицы отдельно. Вы можете иметь несколько реплицированных и несколько не реплицированных таблиц на одном сервере. Вы также можете реплицировать таблицы по-разному, например, одну с двухфакторной репликацией и другую с трехфакторной.

Репликация реализована в движке таблицы ReplicatedMergeTree. Путь в ZooKeeper указывается в качестве параметра движка. Все таблицы с одинаковым путем в ZooKeeper становятся репликами друг друга: они синхронизируют свои данные и поддерживают согласованность. Реплики можно добавлять и удалять динамически, просто создавая или удаляя таблицу.

Репликация использует асинхронную multi-master схему. Вы можете вставить данные в любую реплику, которая имеет открытую сессию в ZooKeeper, и данные реплицируются на все другие реплики асинхронно. Поскольку ClickHouse не поддерживает UPDATE, репликация исключает конфликты (conflict-free replication). Поскольку подтверждение вставок кворумом не реализовано, только что вставленные данные могут быть потеряны в случае сбоя одного узла.

Метаданные для репликации хранятся в ZooKeeper. Существует журнал репликации, в котором перечислены действия, которые необходимо выполнить. Среди этих действий: получить часть (get the part); объединить части (merge parts); удалить партицию (drop a partition) и так далее. Каждая реплика копирует журнал репликации в свою очередь, а затем выполняет действия из очереди. Например, при вставке в журнале создается действие «получить часть» (get the part), и каждая реплика загружает эту часть. Слияния координируются между репликами, чтобы получить идентичные до байта результаты. Все части объединяются одинаково на всех репликах. Одна из реплик-лидеров инициирует новое слияние кусков первой и записывает действия «слияния частей» в журнал. Несколько реплик (или все) могут быть лидерами одновременно. Реплике можно запретить быть лидером с помощью merge\_tree настройки replicated\_can\_become\_leader.

Репликация является физической: между узлами передаются только сжатые части, а не запросы. Слияния обрабатываются на каждой реплике независимо, в большинстве случаев, чтобы снизить затраты на сеть, во избежание усиления роли сети. Крупные объединенные части отправляются по сети только в случае значительной задержки репликации.

Кроме того, каждая реплика сохраняет свое состояние в ZooKeeper в виде набора частей и его контрольных сумм. Когда состояние в локальной файловой системе расходится с эталонным состоянием в ZooKeeper, реплика восстанавливает свою согласованность путем загрузки отсутствующих и поврежденных частей из других реплик. Когда в локальной файловой системе есть неожиданные или испорченные данные, ClickHouse не удаляет их, а перемещает в отдельный каталог и забывает об этом.

### Note

Кластер ClickHouse состоит из независимых шардов, а каждый шард состоит из реплик. Кластер **не является эластичным** (not elastic), поэтому после добавления нового шарда данные не будут автоматически распределены между ними. Вместо этого нужно изменить настройки, чтобы выровнять нагрузку на кластер. Эта реализация дает вам больший контроль, и вполне приемлема для относительно небольших кластеров, таких как десятки узлов. Но для кластеров с сотнями узлов, которые мы используем в эксплуатации, такой подход становится

существенным недостатком. Движки таблиц, которые охватывают весь кластер с динамически реплицируемыми областями, которые могут быть автоматически разделены и сбалансированы между кластерами, еще предстоит реализовать.

# 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.

## Как собрать ClickHouse на Mac OS X

Сборка должна запускаться с `x86_64` (Intel) на macOS версии 10.15 (Catalina) и выше в последней версии компилятора Xcode's native AppleClang, Homebrew's vanilla Clang или в GCC-компиляторах.

## Установка Homebrew

```
$ /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

## Установка Xcode и инструментов командной строки

1. Установите из App Store последнюю версию **Xcode**.
2. Запустите ее, чтобы принять лицензионное соглашение. Необходимые компоненты устанавливаются автоматически.
3. Затем убедитесь, что в системе выбрана последняя версия инструментов командной строки:

```
bash $ sudo rm -rf /Library/Developer/CommandLineTools $ sudo xcode-select --install
```

4. Перезагрузитесь.

## Установка компиляторов, инструментов и библиотек

```
bash $ brew update $ brew install cmake ninja libtool gettext llvm gcc
```

## Просмотр исходников ClickHouse

```
bash $ git clone --recursive git@github.com:ClickHouse/ClickHouse.git # or https://github.com/ClickHouse/ClickHouse.git
```

## Сборка ClickHouse

Чтобы запустить сборку в компиляторе Xcode's native AppleClang:

```
bash $ cd ClickHouse $ rm -rf build $ mkdir build $ cd build $ cmake -DCMAKE_BUILD_TYPE=RelWithDebInfo -DENABLE_JEMALLOC=OFF .. $ cmake --build . --config RelWithDebInfo $ cd ..
```

Чтобы запустить сборку в компиляторе Homebrew's vanilla Clang:

```
bash $ 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 -DENABLE_JEMALLOC=OFF .. $ cmake -DCMAKE_C_COMPILER=$(brew --prefix llvm)/bin/clang -DCMAKE_CXX_COMPILER=$(brew --prefix llvm)/bin/clang++ -DCMAKE_BUILD_TYPE=RelWithDebInfo -DENABLE_JEMALLOC=OFF .. $ cmake --build . --config RelWithDebInfo $ cd ..
```

Чтобы собрать с помощью компилятора Homebrew's vanilla GCC:

```
bash $ 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 -DENABLE_JEMALLOC=OFF .. $ cmake --build . --config RelWithDebInfo $ cd ..
```

## Предупреждения

Если будете запускать `clickhouse-server`, убедитесь, что увеличили системную переменную `maxfiles`.

### Note

Вам понадобится команда `sudo`.

1. Создайте файл `/Library/LaunchDaemons/limit.maxfiles.plist` и поместите в него следующее:

```
``` xml
```

```
Label  
limit.maxfiles  
ProgramArguments
```

```
launchctl  
limit  
maxfiles  
524288  
524288
```

```
RunAtLoad
```

```
ServiceIPC
```

```
```
```

2. Выполните команду:

```
bash $ sudo chown root:wheel /Library/LaunchDaemons/limit.maxfiles.plist
```

3. Перезагрузитесь.

4. Чтобы проверить, как это работает, выполните команду `ulimit -n`.

## 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/troeche/trager/apple-libtapi.git
cd apple-libtapi
INSTALLPREFIX=${CCTOOLS} ./build.sh
./install.sh
cd ..

git clone https://github.com/troeche/trager/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 xJf 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.

## Как писать код на C++

### Общее

1. Этот текст носит рекомендательный характер.
2. Если вы редактируете код, то имеет смысл писать так, как уже написано.
3. Стиль нужен для единства. Единообразие нужно, чтобы было проще (удобнее) читать код. А также, чтобы было легче осуществлять поиск по коду.
4. Многие правила продиктованы не какими либо разумными соображениями, а сложившейся практикой.

## Форматирование

1. Большую часть форматирования сделает автоматически clang-format.
2. Отступы — 4 пробела. Настройте среду разработки так, чтобы таб добавлял четыре пробела.
3. Открывающая и закрывающие фигурные скобки на отдельной строке.

```
inline void readBoolText(bool & x, ReadBuffer & buf)
{
    char tmp = '0';
    readChar(tmp, buf);
    x = tmp != '0';
}
```

4. Если всё тело функции — один statement, то его можно разместить на одной строке. При этом, вокруг фигурных скобок ставятся пробелы (кроме пробела на конце строки).

```
inline size_t mask() const { return buf_size() - 1; }
inline size_t place(HashValue x) const { return x & mask(); }
```

5. Для функций. Пробелы вокруг скобок не ставятся.

```
void reinsert(const Value & x)
```

```
memcpys(&buf[place_value], &x, sizeof(x));
```

6. В выражениях if, for, while и т.д. перед открывающей скобкой ставится пробел (в отличие от вызовов функций).

```
for (size_t i = 0; i < rows; i += storage.index_granularity)
```

**7.** Вокруг бинарных операторов (+, -, \*, /, %, ...), а также тернарного оператора ?: ставятся пробелы.

```
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.** Если ставится перенос строки, то оператор пишется на новой строке, и перед ним увеличивается отступ.

```
if (elapsed_ns)
    message << "("
        << rows_read_on_server * 1000000000 / elapsed_ns << " rows/s., "
        << bytes_read_on_server * 1000.0 / elapsed_ns << " MB/s.) ";
```

**9.** Внутри строки можно, выполнять выравнивание с помощью пробелов.

```
dst.ClickLogID      = click.LogID;
dst.ClickEventID    = click.EventID;
dst.ClickGoodEvent  = click.GoodEvent;
```

**10.** Вокруг операторов , -> не ставятся пробелы.

При необходимости, оператор может быть перенесён на новую строку. В этом случае, перед ним увеличивается отступ.

**11.** Унарные операторы --, ++, \*, &, ... не отделяются от аргумента пробелом.

**12.** После запятой ставится пробел, а перед — нет. Аналогично для точки с запятой внутри выражения for.

**13.** Оператор [] не отделяется пробелами.

**14.** В выражении template <...>, между template и < ставится пробел, а после < и до > не ставится.

```
template <typename TKey, typename TValue>
struct AggregatedStatElement
{}
```

**15.** В классах и структурах, public, private, protected пишется на том же уровне, что и class/struct, а остальной код с отступом.

```
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.** Если на весь файл один namespace и кроме него ничего существенного нет, то отступ внутри namespace не нужен.

**17.** Если блок для выражения if, for, while, ... состоит из одного statement, то фигурные скобки не обязательны. Вместо этого поместите statement на отдельную строку. Это правило справедливо и для вложенных if, for, while, ...

Если внутренний statement содержит фигурные скобки или else, то внешний блок следует писать в фигурных скобках.

```
/// Finish write.  
for (auto & stream : streams)  
    stream.second->finalize();
```

**18.** Не должно быть пробелов на концах строк.

**19.** Исходники в кодировке UTF-8.

**20.** В строковых литералах можно использовать не-ASCII.

```
<< ", " << (timer.elapsed() / chunks_stats.hits) << " µsec(hit.>";
```

**21.** Не пишите несколько выражений в одной строке.

**22.** Внутри функций группируйте блоки кода, отделяя их не более, чем одной пустой строкой.

**23.** Функции, классы, и т. п. отделяются друг от друга одной или двумя пустыми строками.

**24.** `const` (относящийся к значению) пишется до имени типа.

```
//correct  
const char * pos  
const std::string & s  
//incorrect  
char const * pos
```

**25.** При объявлении указателя или ссылки, символы `*` и `&` отделяются пробелами с обеих сторон.

```
//correct  
const char * pos  
//incorrect  
const char* pos  
const char *pos
```

**26.** При использовании шаблонных типов, пишите `using` (кроме, возможно, простейших случаев).

То есть, параметры шаблона указываются только в `using` и затем не повторяются в коде.

`using` может быть объявлен локально, например, внутри функции.

```
//correct  
using FileStreams = std::map<std::string, std::shared_ptr<Stream>>;  
FileStreams streams;  
//incorrect  
std::map<std::string, std::shared_ptr<Stream>> streams;
```

**27.** Нельзя объявлять несколько переменных разных типов в одном выражении.

```
//incorrect  
int x, *y;
```

**28.** C-style cast не используется.

```
//incorrect  
std::cerr << (int)c << std::endl;  
//correct  
std::cerr << static_cast<int>(c) << std::endl;
```

**29.** В классах и структурах, группируйте отдельно методы и отдельно члены, внутри каждой области видимости.

**30.** Для не очень большого класса/структурь, можно не отделять объявления методов от реализации.

Аналогично для маленьких методов в любых классах/структурах.

Для шаблонных классов/структур, лучше не отделять объявления методов от реализации (так как иначе они всё равно должны быть определены в той же единице трансляции).

**31.** Не обязательно умещать код по ширине в 80 символов. Можно в 140.

**32.** Всегда используйте префиксный инкремент/декремент, если постфиксный не нужен.

```
for (Names::const_iterator it = column_names.begin(); it != column_names.end(); ++it)
```

## Комментарии

**1.** Необходимо обязательно писать комментарии во всех нетривиальных местах.

Это очень важно. При написании комментария, можно успеть понять, что код не нужен вообще, или что всё сделано неверно.

```
/** 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.** Комментарии могут быть сколь угодно подробными.

**3.** Комментарии пишутся до соответствующего кода. В редких случаях после, на той же строке.

```
/** 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.** Комментарии следует писать только на английском языке.

**5.** При написании библиотеки, разместите подробный комментарий о том, что это такое, в самом главном заголовочном файле.

**6.** Нельзя писать комментарии, которые не дают дополнительной информации. В частности, нельзя писать пустые комментарии вроде этого:

```
/*
 * 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:
 */
```

Пример взят с ресурса <http://home.tamk.fi/~jaalto/course/coding-style/doc/unmaintainable-code/>.

**7.** Нельзя писать мусорные комментарии (автор, дата создания...) в начале каждого файла.

**8.** Однострочные комментарии начинаются с трёх слешей: `///`, многострочные с `/**`. Такие комментарии считаются «документирующими».

Замечание: такие комментарии могут использоваться для генерации документации с помощью Doxygen. Но, фактически, Doxygen не используется, так как для навигации по коду гораздо удобнее использовать возможности IDE.

**9.** В начале и конце многострочного комментария, не должно быть пустых строк (кроме строки, на которой закрывается многострочный комментарий).

**10.** Для закомментированных кусков кода, используются обычные, не «документирующие» комментарии.

**11.** Удаляйте закомментированные куски кода перед коммитом.

**12.** Не нужно писать нецензурную брань в комментариях или коде.

**13.** Не пишите прописными буквами. Не используйте излишнее количество знаков препинания.

```
/// WHAT THE FAIL???
```

**14.** Не составляйте из комментариев строки-разделители.

```
//
```

**15.** Не нужно писать в комментарии диалог (лучше сказать устно).

```
/// Why did you do this stuff?
```

**16.** Не нужно писать комментарий в конце блока о том, что представлял собой этот блок.

```
/// for
```

## Имена

**1.** В именах переменных и членов класса используйте маленькие буквами с подчёркиванием.

```
size_t max_block_size;
```

**2.** Имена функций (методов) camelCase с маленькой буквы.

```
std::string getName() const override { return "Memory"; }
```

**3.** Имена классов (структур) - CamelCase с большой буквы. Префиксы кроме I для интерфейсов - не используются.

```
class StorageMemory : public IStorage
```

**4.** using называются также, как классы, либо с \_t на конце.

**5.** Имена типов — параметров шаблонов: в простых случаях - T; T, U; T1, T2.

В более сложных случаях - либо также, как имена классов, либо можно добавить в начало букву T.

```
template <typename TKey, typename TValue>
struct AggregatedStatElement
```

**6.** Имена констант — параметров шаблонов: либо также, как имена переменных, либо N в простом случае.

```
template <bool without_www>
struct ExtractDomain
```

**7.** Для абстрактных классов (интерфейсов) можно добавить в начало имени букву I.

```
class IBlockInputStream
```

**8.** Если переменная используется достаточно локально, то можно использовать короткое имя.

В остальных случаях используйте имя, описывающее смысл.

```
bool info_successfully_loaded = false;
```

**9.** В именах define и глобальных констант используется ALL\_CAPS с подчёркиванием.

```
##define MAX_SRC_TABLE_NAMES_TO_STORE 1000
```

**10.** Имена файлов с кодом называйте по стилю соответственно тому, что в них находится.

Если в файле находится один класс, назовите файл, как класс (CamelCase).

Если в файле находится одна функция, назовите файл, как функцию (camelCase).

**11.** Если имя содержит сокращение, то:

- для имён переменных, всё сокращение пишется маленькими буквами mysql\_connection (не mySQL\_connection).
- для имён классов и функций, сохраняются большие буквы в сокращении MySQLConnection (не MySqlConnection).

**12.** Параметры конструктора, использующиеся сразу же для инициализации соответствующих членов класса, следует назвать также, как и члены класса, добавив подчёркивание в конец.

```
FileQueueProcessor(  
    const std::string & path_,  
    const std::string & prefix_,  
    std::shared_ptr<FileHandler> handler_)  
: path(path_),  
prefix(prefix_),  
handler(handler_),  
log(&Logger::get("FileQueueProcessor"))  
{  
}
```

Также можно называть параметры конструктора так же, как и члены класса (не добавлять подчёркивание), но только если этот параметр не используется в теле конструктора.

**13.** Именование локальных переменных и членов класса никак не отличается (никакие префиксы не нужны).

```
timer (not m_timer)
```

**14.** Константы в `enum` — CamelCase с большой буквы. Также допустим ALL\_CAPS. Если `enum` не локален, то используйте `enum class`.

```
enum class CompressionMethod  
{  
    QuickLZ = 0,  
    LZ4     = 1,  
};
```

**15.** Все имена - по-английски. Транслит с русского использовать нельзя.

```
не Stroka
```

**16.** Сокращения (из нескольких букв разных слов) в именах можно использовать только если они являются общепринятыми (если для сокращения можно найти расшифровку в английской википедии или сделав поисковый запрос).

```
`AST`, `SQL`.
```

```
Не `NVDH` (что-то неведомое)
```

Сокращения в виде обрезанного слова можно использовать, только если такое сокращение является широко используемым.

Впрочем, сокращения также можно использовать, если расшифровка находится рядом в комментарии.

**17.** Имена файлов с исходниками на C++ должны иметь расширение только `.cpp`. Заголовочные файлы - только `.h`.

## Как писать код

**1.** Управление памятью.

Ручное освобождение памяти (`delete`) можно использовать только в библиотечном коде.

В свою очередь, в библиотечном коде, оператор `delete` можно использовать только в деструкторах.

В прикладном коде следует делать так, что память освобождается каким-либо объектом, который владеет ей.

Примеры:

- проще всего разместить объект на стеке, или сделать его членом другого класса.
- для большого количества маленьких объектов используйте контейнеры.
- для автоматического освобождения маленького количества объектов, выделенных на куче, используйте `shared_ptr/unique_ptr`.

## 2. Управление ресурсами.

Используйте `RAII` и см. пункт выше.

## 3. Обработка ошибок.

Используйте исключения. В большинстве случаев, нужно только кидать исключения, а ловить - не нужно (потому что `RAII`).

В программах оффлайн обработки данных, зачастую, можно не ловить исключения.

В серверах, обрабатывающих пользовательские запросы, как правило, достаточно ловить исключения на самом верху обработчика соединения.

В функциях потока, следует ловить и запоминать все исключения, чтобы выкинуть их в основном потоке после `join`.

```
/// Если вычислений ещё не было - вычислим первый блок синхронно
if (!started)
{
    calculate();
    started = true;
}
else // Если вычисления уже идут - подождём результата
    pool.wait();

if (exception)
    exception->rethrow();
```

Ни в коем случае не «проглатывайте» исключения без разбора. Ни в коем случае, не превращайте все исключения без разбора в сообщения в логе.

```
//Not correct
catch (...) {}
```

Если вам нужно проигнорировать какие-то исключения, то игнорируйте только конкретные, а остальные кидайте обратно.

```
catch (const DB::Exception & e)
{
    if (e.code() == ErrorCodes::UNKNOWN_AGGREGATE_FUNCTION)
        return nullptr;
    else
        throw;
}
```

При использовании функций, использующих коды возврата или `errno`, проверяйте результат и кидайте исключение.

```
if (0 != close(fd))
    throwFromErrno("Cannot close file " + file_name, ErrorCode::CANNOT_CLOSE_FILE);
```

`assert` не используются.

#### 4. Типы исключений.

В прикладном коде не требуется использовать сложную иерархию исключений. Желательно, чтобы текст исключения был понятен системному администратору.

#### 5. Исключения, вылетающие из деструкторов.

Использовать не рекомендуется, но допустимо.

Используйте следующие варианты:

- Сделайте функцию (`done()` или `finalize()`), которая позволяет заранее выполнить всю работу, в процессе которой может возникнуть исключение. Если эта функция была вызвана, то затем в деструкторе не должно возникать исключений.
- Слишком сложную работу (например, отправку данных по сети) можно вообще не делать в деструкторе, рассчитывая, что пользователь заранее позовёт метод для завершения работы.
- Если в деструкторе возникло исключение, желательно не «проглатывать» его, а вывести информацию в лог (если в этом месте доступен логгер).
- В простых программах, если соответствующие исключения не ловятся, и приводят к завершению работы с записью информации в лог, можно не беспокоиться об исключениях, вылетающих из деструкторов, так как вызов `std::terminate` (в случае `noexcept` по умолчанию в C++11), является приемлемым способом обработки исключения.

#### 6. Отдельные блоки кода.

Внутри одной функции, можно создать отдельный блок кода, для того, чтобы сделать некоторые переменные локальными в нём, и для того, чтобы соответствующие деструкторы были вызваны при выходе из блока.

```
Block block = data.in->read();

{
    std::lock_guard<std::mutex> lock(mutex);
    data.ready = true;
    data.block = block;
}

ready_any.set();
```

#### 7. Многопоточность.

В программах оффлайн обработки данных:

- сначала добейтесь более-менее максимальной производительности на одном процессорном ядре, потом можно распараллеливать код, но только если есть необходимость.

В программах - серверах:

- используйте пул потоков для обработки запросов. На данный момент, у нас не было задач, в которых была бы необходимость использовать userspace context switching.

`Fork` для распараллеливания не используется.

## **8. Синхронизация потоков.**

Часто можно сделать так, чтобы отдельные потоки писали данные в разные ячейки памяти (лучше в разные кэш-линии), и не использовать синхронизацию потоков (кроме `joinAll`).

Если синхронизация нужна, то в большинстве случаев, достаточно использовать `mutex` под `lock_guard`.

В остальных случаях, используйте системные примитивы синхронизации. Не используйте `busy wait`.

Атомарные операции можно использовать только в простейших случаях.

Не нужно писать самостоятельно lock-free структуры данных, если вы не являетесь экспертом.

## **9. Ссылки и указатели.**

В большинстве случаев, предпочтите ссылки.

## **10. `const`.**

Используйте константные ссылки, указатели на константу, `const_iterator`, константные методы.

Считайте, что `const` — вариант написания «по умолчанию», а отсутствие `const` только при необходимости.

Для переменных, передающихся по значению, использовать `const` обычно не имеет смысла.

## **11. `unsigned`.**

Используйте `unsigned`, если нужно.

## **12. Числовые типы.**

Используйте типы `UInt8`, `UInt16`, `UInt32`, `UInt64`, `Int8`, `Int16`, `Int32`, `Int64`, а также `size_t`, `ssize_t`, `ptrdiff_t`.

Не используйте для чисел типы `signed/unsigned long`, `long long`, `short`, `signed/unsigned char`, `char`.

## **13. Передача аргументов.**

Сложные значения передавайте по ссылке (включая `std::string`).

Если функция захватывает владение объектом, созданным на куче, то сделайте типом аргумента `shared_ptr` или `unique_ptr`.

## **14. Возврат значений.**

В большинстве случаев, просто возвращайте значение с помощью `return`. Не пишите `return std::move(res)`.

Если внутри функции создаётся объект на куче и отдаётся наружу, то возвращайте `shared_ptr` или `unique_ptr`.

В некоторых редких случаях, может потребоваться возвращать значение через аргумент функции. В этом случае, аргументом будет ссылка.

```
using AggregateFunctionPtr = std::shared_ptr<IAggregateFunction>;  
/** Позволяет создать агрегатную функцию по её имени.  
 */  
class AggregateFunctionFactory  
{  
public:  
    AggregateFunctionFactory();  
    AggregateFunctionPtr get(const String & name, const DataTypes & argument_types) const;
```

## 15. namespace.

Для прикладного кода отдельный `namespace` использовать не нужно.

Для маленьких библиотек - не требуется.

Для не совсем маленьких библиотек - поместите всё в `namespace`.

Внутри библиотеки в `.h` файле можно использовать `namespace detail` для деталей реализации, не нужных прикладному коду.

В `.cpp` файле можно использовать `static` или анонимный `namespace` для скрытия символов.

Также, `namespace` можно использовать для `enum`, чтобы соответствующие имена не попали во внешний `namespace` (но лучше использовать `enum class`).

## 16. Отложенная инициализация.

Обычно, если для инициализации требуются аргументы, то не пишите конструктор по умолчанию.

Если потом вам потребовалась отложенная инициализация, то вы можете дописать конструктор по умолчанию (который создаст объект с некорректным состоянием). Или, для небольшого количества объектов, можно использовать `shared_ptr/unique_ptr`.

```
Loader(DB::Connection * connection_, const std::string & query, size_t max_block_size_);  
/// Для отложенной инициализации  
Loader() {}
```

## 17. Виртуальные функции.

Если класс не предназначен для полиморфного использования, то не нужно делать функции виртуальными зря. Это относится и к деструктору.

## 18. Кодировки.

Везде используется UTF-8. Используется `std::string, char *`. Не используется `std::wstring, wchar_t`.

## 19. Логирование.

См. примеры везде в коде.

Перед коммитом, удалите всё бессмысленное и отладочное логирование, и другие виды отладочного вывода.

Не должно быть логирования на каждую итерацию внутреннего цикла, даже уровня `Trace`.

При любом уровне логирования, логи должно быть возможно читать.

Логирование следует использовать, в основном, только в прикладном коде.

Сообщения в логе должны быть написаны на английском языке.

Желательно, чтобы лог был понятен системному администратору.

Не нужно писать ругательства в лог.

В логе используется кодировка UTF-8. Изредка можно использовать в логе не-ASCII символы.

## 20. Ввод-вывод.

Во внутренних циклах (в критичных по производительности участках программы) нельзя использовать `iostreams` (в том числе, ни в коем случае не используйте `stringstream`).

Вместо этого используйте библиотеку `DB/IO`.

## 21. Дата и время.

См. библиотеку `DateLUT`.

## 22. include.

В заголовочном файле используется только `#pragma once`, а `include guards` писать не нужно.

## 23. using.

`using namespace` не используется. Можно использовать `using` что-то конкретное. Лучше локально, внутри класса или функции.

## 24. Не нужно использовать trailing return type для функций, если в этом нет необходимости.

```
auto f() -> void
```

## 25. Объявление и инициализация переменных.

```
//right way
std::string s = "Hello";
std::string s{"Hello"};

//wrong way
auto s = std::string{"Hello"};
```

## 26. Для виртуальных функций, пишите `virtual` в базовом классе, а в классах-наследниках, пишите `override` и не пишите `virtual`.

# Неиспользуемые возможности языка C++

## 2. Спецификаторы исключений из C++03 не используются.

# Сообщения об ошибках

Сообщения об ошибках -- это часть пользовательского интерфейса программы, предназначенная для того, чтобы позволить пользователю:

- замечать ошибочные ситуации,
- понимать их смысл и причины,
- устранять эти ситуации.

Форма и содержание сообщений об ошибках должны способствовать достижению этих целей.

Есть два основных вида ошибок:

- пользовательская или системная ошибка,
- внутренняя программная ошибка.

## Пользовательская ошибка

Такая ошибка вызвана действиями пользователя (неверный синтаксис запроса) или конфигурацией внешних систем (кончилось место на диске). Предполагается, что пользователь может устранить её самостоятельно. Для этого в сообщении об ошибке должна содержаться следующая информация:

- что произошло. Это должно объясняться в пользовательских терминах (`Function pow() is not supported for data type UInt128`), а не загадочными конструкциями из кода (`runtime overload resolution failed in DB::BinaryOperationBuilder<FunctionAdaptor<pow>::Impl, UInt128, Int8>::kaboolgleFastPath()`).
- почему/где/когда -- любой контекст, который помогает отладить проблему. Представьте, как бы её отлаживали вы (программировать и пользоваться отладчиком нельзя).
- что можно предпринять для устранения ошибки. Здесь можно перечислить типичные причины проблемы, настройки, влияющие на это поведение, и так далее.

Пример нормального сообщения:

```
No alias for subquery or table function in JOIN (set joined_subquery_requires_alias=0 to disable restriction).
While processing '(SELECT 2 AS a)'.
```

Сказано что не хватает алиаса, показано, для какой части запроса, и предложена настройка, позволяющая ослабить это требование.

Пример катастрофически плохого сообщения:

```
The dictionary is configured incorrectly.
```

Из него не понятно:

- какой словарь?
- в чём ошибка конфигурации?

Что может сделать пользователь в такой ситуации: применять внешние отладочные инструменты, спрашивать совета на форумах, гадать на кофейной гуще, и, конечно же, ненавидеть софт, который над ним так издевается. Не нужно издеваться над пользователями, это плохой UX.

## Внутренняя программная ошибка

Такая ошибка вызвана нарушением внутренних инвариантов программы: например, внутренняя функция вызвана с неверными параметрами, не совпадают размеры колонок в блоке, произошло разыменование нулевого указателя, и так далее. Сигналы типа SIGSEGV относятся к этой же категории.

Появление такой ошибки всегда свидетельствует о наличии бага в программе. Пользователь не может исправить такую ошибку самостоятельно, и должен сообщить о ней разработчикам.

Есть два основных варианта проверки на такие ошибки:

- Исключение с кодом `LOGICAL_ERROR`. Его можно использовать для важных проверок, которые делаются в том числе в релизной сборке.

- `assert`. Такие условия не проверяются в релизной сборке, можно использовать для тяжёлых и опциональных проверок.

Пример сообщения, у которого должен быть код `LOGICAL_ERROR`:

Block header is inconsistent with Chunk in IComplicatedProcessor::munge(). It is a bug!

По каким признакам можно заметить, что здесь говорится о внутренней программной ошибке?

- в сообщении упоминаются внутренние сущности из кода,
- в сообщении написано it's a bug,
- непосредственные действия пользователя не могут исправить эту ошибку. Мы ожидаем, что пользователь зарепортит её как баг, и будем исправлять в коде.

## Как выбрать код ошибки?

Код ошибки предназначен для автоматической обработки некоторых видов ошибок, подобно кодам HTTP. SQL стандартизирует некоторые коды, но на деле ClickHouse не всегда соответствует этим стандартам. Лучше всего выбрать существующий код из `ErrorCodes.cpp`, который больше всего подходит по смыслу. Можно использовать общие коды типа `BAD_ARGUMENTS` или `TYPE_MISMATCH`. Заводить новый код нужно, только если вы чётко понимаете, что вам нужна специальная автоматическая обработка конкретно этой ошибки на клиенте. Для внутренних программных ошибок используется код `LOGICAL_ERROR`.

## Как добавить новое сообщение об ошибке?

Когда добавляете сообщение об ошибке:

1. Опишите, что произошло, в пользовательских терминах, а не кусками кода.
2. Добавьте максимум контекста (с чем произошло, когда, почему, и т.д.).
3. Добавьте типичные причины.
4. Добавьте варианты исправления (настройки, ссылки на документацию).
5. Вообразите дальнейшие действия пользователя. Ваше сообщение должно помочь ему решить проблему без использования отладочных инструментов и без чужой помощи.
6. Если сообщение об ошибке не формулируется в пользовательских терминах, и действия пользователя не могут исправить проблему -- это внутренняя программная ошибка, используйте код `LOGICAL_ERROR` или `assert`.

## Платформа

**1.** Мы пишем код под конкретные платформы.

Хотя, при прочих равных условиях, предпочтается более-менее кроссплатформенный или легко портируемый код.

**2.** Язык - C++20 (см. список доступных [C++20 фич](#)).

**3.** Компилятор - `clang`. На данный момент (апрель 2021), код собирается версией 11. (Также код может быть собран `gcc` версии 10, но такая сборка не тестируется и непригодна для продакшена).

Используется стандартная библиотека (реализация `libc++`).

**4.** ОС - Linux, Mac OS X или FreeBSD.

**5.** Код пишется под процессоры с архитектурой `x86_64`, `AArch64` и `ppc64le`.

**6.** Используются флаги компиляции `-Wall` `-Wextra` `-Werror` и `-Weverything` с некоторыми исключениями.

**7.** Используется статическая линковка со всеми библиотеками кроме `libc`.

# Инструментарий

1. Хорошая среда разработки - KDevelop.
2. Для отладки используется `gdb`, `valgrind` (`memcheck`), `strace`, `-fsanitize=...`, `tcmalloc_minimal_debug`.
3. Для профилирования используется `Linux Perf`, `valgrind` (`callgrind`), `strace -cf`.
4. Исходники в `Git`.
5. Сборка с помощью `CMake`.
6. Программы выкладываются с помощью `deb` пакетов.
7. Коммиты в `master` не должны ломать сборку проекта.

А работоспособность собранных программ гарантируется только для отдельных ревизий.

8. Коммите как можно чаще, в том числе и нерабочий код.

Для этого следует использовать бранчи.

Если ваш код в ветке `master` ещё не собирается, исключите его из сборки перед `push`, также вы будете должны его доработать или удалить в течение нескольких дней.

9. Для нетривиальных изменений, используются бранчи. Следует загружать бранчи на сервер.
10. Ненужный код удаляется из исходников.

## Библиотеки

1. Используются стандартные библиотеки C++20 (допустимо использовать экспериментальные расширения), а также фреймворки `boost`, `Poco`.
2. Библиотеки должны быть расположены в виде исходников в директории `contrib` и собираться вместе с ClickHouse. Не разрешено использовать библиотеки, доступные в пакетах ОС, или любые другие способы установки библиотек в систему. Подробнее смотрите раздел [Рекомендации по добавлению сторонних библиотек и поддержанию в них пользовательских изменений](#).
3. Предпочтение отдается уже использующимся библиотекам.

## Общее

1. Пишите как можно меньше кода.
2. Пробуйте самое простое решение.
3. Не нужно писать код, если вы ещё не знаете, что будет делать ваша программа, и как будет работать её внутренний цикл.
4. В простейших случаях, используйте `using` вместо классов/структур.
5. Если есть возможность - не пишите конструкторы копирования, операторы присваивания, деструктор (кроме виртуального, если класс содержит хотя бы одну виртуальную функцию), move-конструкторы и move-присваивания. То есть, чтобы соответствующие функции, генерируемые компилятором, работали правильно. Можно использовать `default`.
6. Приветствуется упрощение и уменьшение объёма кода.

## Дополнительно

1. Явное указание `std::` для типов из `stddef.h`.

Рекомендуется не указывать. То есть, рекомендуется писать `size_t` вместо `std::size_t`, это короче.

При желании, можно дописать `std::`, этот вариант допустим.

## 2. Явное указание `std::` для функций из стандартной библиотеки C.

Не рекомендуется. То есть, пишите `memcpuy` вместо `std::memcpuy`.

Причина - существуют похожие нестандартные функции, например, `memmem`. Мы можем использовать и изредка используем эти функции. Эти функции отсутствуют в namespace `std`.

Если вы везде напишете `std::memcpuy` вместо `memcpuy`, то будет неудобно смотреться `memmem` без `std::`.

Тем не менее, указывать `std::` тоже допустимо, если так больше нравится.

## 3. Использование функций из C при наличии аналогов в стандартной библиотеке C++.

Допустимо, если это использование эффективнее.

Для примера, для копирования длинных кусков памяти, используйте `memcpuy` вместо `std::copy`.

## 4. Перенос длинных аргументов функций.

Допустимо использовать любой стиль переноса, похожий на приведённые ниже:

```
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 requires 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 `uncrustify` tool to reformat your code. Configuration is in `uncrustify.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.

## Используемые сторонние библиотеки

Список сторонних библиотек:

| Библиотека | Тип лицензии |
|------------|--------------|
| abseil-cpp | Apache       |
| AMQP-CPP   | Apache       |

| Библиотека         | Тип лицензии |
|--------------------|--------------|
| arrow              | Apache       |
| avro               | Apache       |
| aws                | Apache       |
| aws-c-common       | Apache       |
| aws-c-event-stream | Apache       |
| aws-checksums      | Apache       |
| base64             | BSD 2-clause |
| boost              | Boost        |
| 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       |

| Библиотека          | Тип лицензии  |
|---------------------|---------------|
| googletest          | BSD 3-clause  |
| grpc                | Apache        |
| h3                  | Apache        |
| hyperscan           | Boost         |
| icu                 | Public Domain |
| icudata             | Public Domain |
| jemalloc            | BSD 2-clause  |
| krb5                | MIT           |
| 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          |

| Библиотека     | Тип лицензии  |
|----------------|---------------|
| miniselect     | Boost         |
| msgpack-c      | Boost         |
| murmurhash     | Public Domain |
| NuRaft         | Apache        |
| openldap       | Unknown       |
| orc            | Apache        |
| poco           | Boost         |
| protobuf       | BSD 3-clause  |
| 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           |

Список всех сторонних библиотек можно получить с помощью запроса:

```
SELECT library_name, license_type, license_path FROM system.licenses ORDER BY library_name COLLATE 'en';
```

## Пример

# Рекомендации по добавлению сторонних библиотек и поддержанию в них пользовательских изменений

1. Весь внешний сторонний код должен находиться в отдельных папках внутри папки contrib репозитория ClickHouse. По возможности, используйте сабмодули Git.
2. Клонируйте официальный репозиторий [Clickhouse-extras](#). Используйте официальные репозитории GitHub, если они доступны.
3. Создавайте новую ветку на основе той ветки, которую вы хотите интегрировать: например, master -> clickhouse/master или release/vX.Y.Z -> clickhouse/release/vX.Y.Z.
4. Все копии [Clickhouse-extras](#) можно автоматически синхронизировать с удаленными репозиториями. Ветки clickhouse/... останутся незатронутыми, поскольку скорее всего никто не будет использовать этот шаблон именования в своих репозиториях.
5. Добавьте сабмодули в папку contrib репозитория ClickHouse, на который ссылаются клонированные репозитории. Настройте сабмодули для отслеживания изменений в соответствующих ветках clickhouse/....
6. Каждый раз, когда необходимо внести изменения в код библиотеки, следует создавать отдельную ветку, например clickhouse/my-fix. Затем эта ветка должна быть слита (merge) в ветку, отслеживаемую сабмодулем, например, в clickhouse/master или clickhouse/release/vX.Y.Z.
7. Не добавляйте код в клоны репозитория [Clickhouse-extras](#), если имя ветки не соответствует шаблону clickhouse/....
8. Всегда вносите изменения с учетом того, что они попадут в официальный репозиторий. После того как PR будет влит из (ветки разработки/исправлений) вашего личного клона репозитория в [Clickhouse-extras](#), и сабмодуль будет добавлен в репозиторий ClickHouse, рекомендуется сделать еще один PR из (ветки разработки/исправлений) репозитория [Clickhouse-extras](#) в официальный репозиторий библиотеки. Таким образом будут решены следующие задачи: 1) публикуемый код может быть использован многократно и будет иметь более высокую ценность; 2) другие пользователи также смогут использовать его в своих целях; 3) поддержкой кода будут заниматься не только разработчики ClickHouse.
9. Чтобы сабмодуль начал использовать новый код из исходной ветки (например, master), сначала следует аккуратно выполнить слияние (master -> clickhouse/master), и только после этого изменения могут быть добавлены в основной репозиторий ClickHouse. Это связано с тем, что в отслеживаемую ветку (например, clickhouse/master) могут быть внесены изменения, и поэтому ветка может отличаться от первоисточника (master).

## Навигация по коду ClickHouse

Для навигации по коду онлайн доступен [Woboq](#), он расположен [здесь](#). В нём реализовано удобное перемещение между исходными файлами, семантическая подсветка, подсказки, индексация и поиск. Слепок кода обновляется ежедневно.

Также вы можете просматривать исходники на [GitHub](#).

Если вы интересуетесь, какую среду разработки выбрать для работы с ClickHouse, мы рекомендуем CLion, QT Creator, VSCode или KDevelop (с некоторыми предостережениями). Вы можете использовать свою любимую среду разработки, Vim и Emacs тоже считаются.

## 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                                                                                                                   |
|-------------------------------------------|------------------------------------|-----------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------|
| <code>ENABLE_CLICKHOUSE_ALL</code>        | <code>ON</code>                    | Enable all ClickHouse modes by default                          | The clickhouse tool multiple executors may be built automatically know what mode SERVER and CLIENT                        |
| <code>ENABLE_CLICKHOUSE_BENCHMARK</code>  | <code>ENABLE_CLICKHOUSE_ALL</code> | Queries benchmarking mode                                       | <a href="https://clickhouse.com/docs/en/interfaces/benchmark/">https://clickhouse.com/docs/en/interfaces/benchmark/</a>   |
| <code>ENABLE_CLICKHOUSE_CLIENT</code>     | <code>ENABLE_CLICKHOUSE_ALL</code> | Client mode (interactive tui/shell that connects to the server) |                                                                                                                           |
| <code>ENABLE_CLICKHOUSE_COMPRESSOR</code> | <code>ENABLE_CLICKHOUSE_ALL</code> | Data compressor and decompressor                                | <a href="https://clickhouse.com/docs/en/interfaces/compressor/">https://clickhouse.com/docs/en/interfaces/compressor/</a> |
| <code>ENABLE_CLICKHOUSE_COPIER</code>     | <code>ENABLE_CLICKHOUSE_ALL</code> | Inter-cluster data copying mode                                 | <a href="https://clickhouse.com/docs/en/interfaces/copier/">https://clickhouse.com/docs/en/interfaces/copier/</a>         |

| Name                                  | Default value         | Description                                                                         | Comment                                                           |
|---------------------------------------|-----------------------|-------------------------------------------------------------------------------------|-------------------------------------------------------------------|
| 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://present">https://present</a>                     |
| ENABLE_CLICKHOUSE_INSTALL             | OFF                   | Install ClickHouse without .deb/.rpm/.tgz packages (having the binary only)         |                                                                   |
| 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://clickhouse.local/">https://clickhouse.local/</a> |

| Name                                         | Default value         | Description                                                                    | Comment                                                                                             |
|----------------------------------------------|-----------------------|--------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------|
| ENABLE_CLICKHOUSE_OBFUSCATOR                 | ENABLE_CLICKHOUSE_ALL | Table data obfuscator (convert real data to benchmark-ready one)               | <a href="https://clickhouse.com/docs/en/obfuscator/">https://clickhouse.com/docs/en/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 using Avro serialization. |
| 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                       |                                       |

| Name                      | Default value                    | Description                                                               | Comment                                             |
|---------------------------|----------------------------------|---------------------------------------------------------------------------|-----------------------------------------------------|
| 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://ccach">https://ccach</a>           |
| ENABLE_CLANG_TIDY         | OFF                              | Use clang-tidy static analyzer                                            | <a href="https://clang-tidy">https://clang-tidy</a> |
| ENABLE_CURL               | ENABLE_LIBRARIES                 | Enable curl                                                               |                                                     |
| ENABLE_DATASKETCHES       | ENABLE_LIBRARIES                 | Enable DataSketches                                                       |                                                     |
| 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                                                      |                                                     |

| Name               | Default value    | Description                          | Comment |
|--------------------|------------------|--------------------------------------|---------|
| 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 | Enalbe 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 |         |
| 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                        |         |

| Name         | Default value    | Description                       | Comment                                          |
|--------------|------------------|-----------------------------------|--------------------------------------------------|
| 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                                  |                                                     |
| 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 systems have brotly library bundled by default |
| USE_INTERNAL_CAPNP_LIBRARY  | NOT_UNBUNDLED        | Set to FALSE to use system capnproto library instead of bundled                             |                                                     |

| Name                              | Default value | Description                                                                                                | Comment                                                                                                              |
|-----------------------------------|---------------|------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------|
| 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.<br>(Experimental. Set to OFF on your own risk) | Normally w framework USE_INTER OFF to forc gRPC fram installed in The extern be installed running su libgrpc++- grpc |
| USE_INTERNAL_GTEST_LIBRARY        | NOT_UNBUNDLED | Set to FALSE to use system Google Test instead of bundled                                                  |                                                                                                                      |
| 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<br>(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                                                  |                                                                                                                      |

| Name                          | Default value                       | Description                                                                                  | Comment                            |
|-------------------------------|-------------------------------------|----------------------------------------------------------------------------------------------|------------------------------------|
| 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_LIBGSASL_LIBRARY | USE_STATIC_LIBRARIES                | Set to FALSE to use system libgsasl library instead of bundled                               | when USE_ usually needs dependency |
| 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                                |                                    |
| 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) |                                    |

| Name                           | Default value | Description                                                                                            | Comment                                                                                                           |
|--------------------------------|---------------|--------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------|
| 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.<br>(Experimental. Set to OFF on your own risk) | Normally w USE_INTERNAL_TO OFF to force protobuf lib installed in The extern installed in sudo apt-get protobuf-c |
| 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                                           |                                                                                                                   |
| 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                                           |                                                                                                                   |

| Name                            | Default value | Description                                                      | Comment |
|---------------------------------|---------------|------------------------------------------------------------------|---------|
| 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 lld is used |

| Name                      | Default value | Description                                                                                                                                                                                                           | Comment                                                                                                                                                                                                                                                                    |
|---------------------------|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ARCH_NATIVE               | 0             | Add -march=native compiler flag. This makes your binaries non-portable but more performant code may be generated. This option overrides ENABLE_* 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 /tmp usage, mc                                                                                                                                                                                                                                                        |
| ENABLE_CHECK_HEAVY_BUILDS | OFF           | Don't allow C++ translation units to compile too long or to take too much memory while compiling.                                                                                                                     | Take care to add pr thinks that prlimit is to work with multipl 31T18:06:32.65532 as=10000000000 --..... std=gnu++2a .src/CMakeFiles/dbm MF src/CMakeFiles/dbm -o src/CMakeFiles/dbm c ..../src/Storages/Mer 31T18:06:32.65670 ..../src/Storages/Merg to use --ccache-skip |

| Name                                    | Default value        | Description                                                                                                          | Comment                                                                                                                   |
|-----------------------------------------|----------------------|----------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------|
| ENABLE_EXAMPLES                         | OFF                  | Build all example programs in 'examples' subdirectories                                                              |                                                                                                                           |
| ENABLE_FUZZING                          | OFF                  | Fuzzy testing using libfuzzer                                                                                        |                                                                                                                           |
| ENABLE_LIBRARIES                        | ON                   | Enable all external libraries by default                                                                             | Turns on all external libraries by default.                                                                               |
| ENABLE_MULTITARGET_CODE                 | ON                   | Enable platform-dependent code                                                                                       | ClickHouse development macro (e.g. <code>ifdef ENABLE_LIBRARY</code> ) such macro. See <code>src/Makefile</code> .        |
| ENABLE_TESTS                            | ON                   | Provide unit_test_dbms target with Google.Test unit tests                                                            | If turned ON, assumes bundled one.                                                                                        |
| ENABLE_THINLTO                          | ON                   | Clang-specific link time optimization                                                                                | <a href="https://clang.llvm.org/docs/ThinLTO.html">https://clang.llvm.org/docs/ThinLTO.html</a> when building with Clang. |
| 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. with <code>cmake -DENABLE_LIBRARY=OFF</code> the CMake will continue to build.                        |
| GLIBC_COMPATIBILITY                     | ON                   | Enable compatibility with older glibc libraries.                                                                     | Only for Linux, x86_64.                                                                                                   |
| LINKER_NAME                             | OFF                  | Linker name or full path                                                                                             | Example values: <code>lld-link</code> , <code>ld</code> .                                                                 |
| MAKE_STATIC_LIBRARIES                   | USE_STATIC_LIBRARIES | Disable to make shared libraries                                                                                     |                                                                                                                           |

| Name                          | Default value     | Description                                                           | Comment                                                                                                                     |
|-------------------------------|-------------------|-----------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| 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: - <code>ad</code> undefined (UBSan) - ' <code>'</code>                                                     |
| 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 (but debug some source variables)."                                                                 |
| UNBUNDLED                     | OFF               | Use system libraries instead of ones in contrib/                      | We recommend avc can't guarantee all i exists for enthusiasts 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/ClickHouse/use_include_what_you_use">https://github.com/ClickHouse/use_include_what_you_use</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                                                            |                                                                                                                             |

| Name                 | Default value    | Description                                      | Comment                                                        |
|----------------------|------------------|--------------------------------------------------|----------------------------------------------------------------|
| 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 cause expansion).                        |
| WEVERYTHING          | ON               | Enable -Weverything option with some exceptions. | Add some warnings. Wpedantic. Intended to be found useful. API |
| 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.
    ${STRIP_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

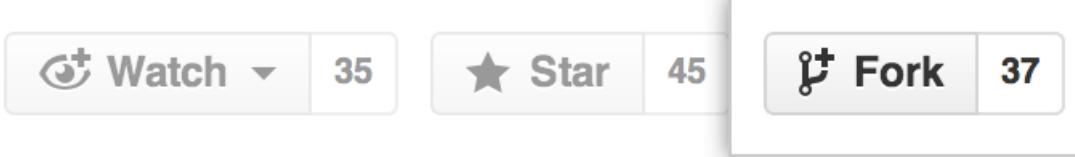
1) [create GitHub account](#) (if you haven't one yet)

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.

## Поставщики облачных услуг ClickHouse

### 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.

## Коммерческие услуги

Данный раздел содержит описание коммерческих услуг, предоставляемых для ClickHouse. Поставщики этих услуг — независимые компании, которые могут не быть аффилированы с Яндексом.

Категории услуг:

- [Облачные услуги](#)
- [Поддержка](#)

## Общие вопросы о ClickHouse

Вопросы:

- [Что такое ClickHouse?](#)
- [Почему ClickHouse такой быстрый?](#)
- [Кто пользуется ClickHouse?](#)
- [Что обозначает название ClickHouse?](#)
- [Как фраза “Не тормозит” осталась на всех футболках?](#)
- [Что такое OLAP?](#)
- [Что такое столбцовая база данных?](#)
- [Почему бы не использовать системы типа MapReduce?](#)

### Если вы не нашли то, что искали:

Загляните в другие категории F.A.Q. или поищите в остальных разделах документации, ориентируясь по оглавлению слева.

## Почему ClickHouse так быстро работает?

Производительность изначально заложена в архитектуре ClickHouse. Высокая скорость выполнения запросов была и остается самым важным критерием, который учитывается при разработке. Но мы обращаем внимание и на другие характеристики, такие как удобство использования, масштабируемость, безопасность. Всё это делает ClickHouse настоящей промышленной разработкой.

Сначала ClickHouse создавался как прототип, который должен был отлично справляться с одной единственной задачей — отбирать и агрегировать данные с максимальной скоростью. Это необходимо, чтобы создать обычный аналитический отчет, и именно это делает стандартный запрос [GROUP BY](#). Для решения такой задачи команда разработки ClickHouse приняла несколько архитектурных решений:

### Столбцовое хранение данных

Исходные данные часто содержат сотни или даже тысячи столбцов, в то время как для конкретного отчета нужны только несколько из них. Система не должна читать ненужные столбцы, поскольку операции чтения данных с диска — самые дорогостоящие.

### Индексы

ClickHouse хранит структуры данных в оперативной памяти, что позволяет считывать не только нужные столбцы, но и нужные диапазоны строк для этих столбцов.

### Сжатие данных

Различные способы хранения смежных значений в столбце позволяют достигать более высокой степени сжатия данных (по сравнению с обычными строковыми СУБД), т.к. в смежных строках значения часто бывают одинаковыми или близкими. В дополнение к универсальному сжатию

ClickHouse поддерживает [специализированные кодеки](#), которые позволяют еще больше уменьшить объемы хранимых данных.

## Векторные запросы

ClickHouse не только хранит, но и обрабатывает данные в столбцах. Это приводит к лучшей утилизации кеша процессора и позволяет использовать инструкции [SIMD](#).

## Масштабируемость

ClickHouse может задействовать все доступные мощности процессоров и объемы дисков, чтобы выполнить даже одиночный запрос. Не только на отдельном сервере, но и в целом кластере.

Похожие техники используют и многие другие СУБД. **Внимание к мельчайшим деталям** — вот что на самом деле выделяет ClickHouse. Большинство языков программирования поддерживают большинство распространенных алгоритмов и структур данных, но как правило, они бывают слишком универсальными, чтобы быть по-настоящему эффективными. Мы рассматриваем каждую задачу как тонкий инструмент со множеством настроек, вместо того чтобы просто взять какую-то случайную реализацию. Например, если вам нужна хеш-таблица, вот несколько ключевых вопросов, которые нужно продумать:

- Какую хеш-функцию выбрать?
- Каким способом разрешать коллизии: [открытая адресация](#) или [метод цепочек](#)?
- Как хранить данные в памяти: ключи и значения в одном массиве или в отдельных? Будут ли там храниться маленькие или большие значения?
- Фактор заполнения: когда и как менять размер таблицы? Как перемещать значения при изменении размера?
- Будут ли значения удаляться и если да, то какой алгоритм сработает лучше?
- Понадобится ли быстрое зондирование с использованием битовых масок, встроенное хранение строковых ключей, поддержка неперемещаемых значений, предварительная выборка и пакетная обработка?

Хеш-таблица — ключевая структура данных для реализации `GROUP BY`, и ClickHouse автоматически выбирает одну из [более 30 вариаций](#) для каждого специфического запроса.

Для алгоритмов сортировки, например, следует продумать следующие вопросы:

- Что будет сортироваться: массив чисел, кортежей, строк или структур?
- Доступны ли все данные в оперативной памяти?
- Нужна ли стабильная сортировка?
- Нужна ли полная сортировка? Может быть, будет достаточно частичной или выборочной сортировки?
- Как сравнивать данные?
- Не являются ли данные частично отсортированными?

Алгоритмы, основанные на характеристиках рабочих данных, обычно дают лучшие результаты, чем их более универсальные аналоги. Если заранее неизвестно, с какими данными придется работать, ClickHouse будет в процессе выполнения пробовать различные реализации и в итоге выберет оптимальный вариант. Например, рекомендуем прочитать [статью о том, как в ClickHouse реализуется распаковка LZ4](#).

Ну и последнее, но тем не менее важное условие: команда ClickHouse постоянно отслеживает в интернете сообщения пользователей о найденных ими удачных реализациях, алгоритмах или структурах данных, анализирует и пробует новые идеи. Иногда в этом потоке сообщений попадаются действительно ценные предложения.

## Советы о том, как создать собственную высокопроизводительную систему

- При проектировании системы обращайте внимание на мельчайшие детали реализации.
- Учитывайте возможности аппаратного обеспечения.
- Выбирайте структуры и представления данных исходя из требований конкретной задачи.
- Для особых случаев разрабатывайте специализированные решения.
- Пробуйте новые алгоритмы, о которых вы вчера прочитали в интернете. Ищите возможности для совершенствования.
- Выбирайте алгоритмы динамически, в процессе выполнения, на основе статистики.
- Ориентируйтесь на показатели, собранные при работе с реальными данными.
- Проверяйте производительность в процессе CI.
- Измеряйте и анализируйте всё, что только возможно.

## Кто пользуется ClickHouse?

Так как СН является продуктом с открытым исходным кодом, на этот вопрос не так просто ответить. Вы не должны сообщать кому-либо о том, что вы начали пользоваться ClickHouse, достаточно взять исходный код или предкомпилированный установочный пакет. Не нужно подписывать контракт, а [лицензия Apache 2.0](#) позволяет распространять ПО без ограничений.

Кроме того, стек используемых технологий часто не раскрывается из-за NDA. Некоторые компании рассматривают технологии, которыми пользуются, как своё конкурентное преимущество, даже если это продукты с открытым исходным кодом. Такие компании не позволяют сотрудникам рассказывать о том, с каким ПО они работают, или требуют согласовывать это с PR-отделом.

Итак, как же узнать, кто пользуется ClickHouse?

Один из способов — **поспрашивать в своем окружении**. В разговорах люди более охотно делятся тем, какие технологии внедрены в их компаниях, какие задачи решаются с их помощью, могут назвать характеристики аппаратного обеспечения, объемы данных и т.д. Мы регулярно разговариваем с пользователями во время [митапов ClickHouse](#) по всему миру и слышали о более чем 1000 компаний, которые пользуются ClickHouse. К сожалению, мы не можем раскрывать подробности, потому что по умолчанию считаем такие истории защищенными NDA, чтобы избежать любых возможных проблем. Вы можете прийти на любой из наших будущих митапов и самостоятельно поговорить с другими пользователями. Мы анонсируем события по разным каналам, например, вы можете подписаться на [наш Twitter](#).

Второй способ узнать — посмотреть, что компании **говорят публично** о том, как именно они пользуются ClickHouse. Это более существенная информация, потому что ее можно найти в публикациях в блогах, видеозаписях разговоров, презентациях и т.д. Мы собираем ссылки на такие материалы на своей странице **Пользователи ClickHouse**. Будем рады, если вы поделитесь историей вашей компании или ссылками по теме (но всегда помните о том, что не стоит нарушать NDA).

В числе пользователей есть множество очень крупных компаний, знакомых вам, таких как Bloomberg, Cisco, China Telecom, Tencent или Uber, но на самом деле это далеко не полный перечень. К примеру, если вы возьмете **список Forbes крупнейших ИТ-компаний в 2020**, то увидите, что более половины из этих компаний так или иначе пользуются ClickHouse. Также необходимо упомянуть Яндекс — компанию, которая открыла исходный код ClickHouse в 2016 году и является одной из самых крупных ИТ-компаний в Европе.

## Что означает название ClickHouse?

Это комбинация терминов **Clickstream** и **Data wareHouse**. Название пришло из Яндекс.Метрики, для которой первоначально был разработан ClickHouse — там он использовался для хранения истории визитов пользователей на сайты и всех пользовательских действий — "кликов". Кстати, ClickHouse по-прежнему выполняет эту функцию. Узнать об этом больше можно на странице **истории ClickHouse**.

Поскольку название составное, использовать его нужно следующим образом:

- единственно правильный способ написания — ClickHouse — с заглавной буквой H;
- если нужно сокращенное название, используйте **СН**. Исторически сложилось, что в Китае также популярно сокращение СК — в основном, из-за того, что это название использовалось в одном из первых обсуждений ClickHouse на китайском языке.

### Забавный факт

Спустя годы после того, как ClickHouse получил свое название, принцип комбинирования двух слов, каждое из которых имеет подходящий смысл, был признан лучшим способом назвать базу данных в **исследовании Andy Pavlo**, Associate Professor of Databases в Carnegie Mellon University. ClickHouse разделил награду "за лучшее название СУБД" с Postgres.

## Что значит “Не тормозит”?

Обычно этот вопрос возникает, когда люди видят официальные футболки ClickHouse. На них большими буквами написано **“ClickHouse не тормозит”**.

До того, как код ClickHouse стал открытым, его разрабатывали как собственную систему хранения данных в крупнейшей российской ИТ-компании **Яндекс**. Поэтому оригинальный слоган написан по-русски. После выхода версии с открытым исходным кодом мы впервые выпустили некоторое количество таких футболок для мероприятий в России, и просто оставили прежний слоган.

Когда мы решили отправить партию этих футбольок на мероприятия вне России, мы пробовали подобрать подходящий английский слоган. К сожалению, мы так и не смогли придумать достаточно точный и выразительный перевод, ведь на русском этот слоган звучит очень ёмко и при этом довольно элегантно. К тому же, существовало ограничение по количеству символов на

футболках. В итоге мы решили оставить русский вариант даже для международных событий. И это стало прекрасным решением, потому что люди по всему миру приятно удивлялись, когда видели фразу и интересовались, что же там написано.

Итак, как же объяснить эту фразу на английском? Вот несколько вариантов:

- Если переводить буквально, то получится что-то подобное: "*ClickHouse doesn't press the brake pedal*".
- Если же вы хотите максимально сохранить том смысл, который вкладывает в эту фразу человек из ИТ-сферы, то будет примерно следующее: "*If your larger system lags, it's not because it uses ClickHouse*".
- Более короткие, но не такие точные версии: "*ClickHouse is not slow*", "*ClickHouse doesn't lag*" или просто "*ClickHouse is fast*".

Если вы не видели наших футболок, посмотрите видео о ClickHouse. Например, вот это:

P.S. Эти футболки не продаются, а распространяются бесплатно на большинстве митапов [ClickHouse](#), обычно в награду за самые интересные вопросы или другие виды активного участия.

## Что такое OLAP?

**OLAP** (OnLine Analytical Processing) переводится как обработка данных в реальном времени. Это широкий термин, который можно рассмотреть с двух сторон: с технической и с точки зрения бизнеса. Для самого общего понимания можно просто прочитать его с конца:

### **Processing**

Обрабатываются некие исходные данные...

### **Analytical**

... чтобы получить какие-то аналитические отчеты или новые знания...

### **OnLine**

... в реальном времени, практически без задержек на обработку.

# OLAP с точки зрения бизнеса

В последние годы бизнес-сообщество стало осознавать ценность данных. Компании, которые принимают решения вслепую, чаще всего отстают от конкурентов. Управление бизнесом на основе данных, которое применяется успешными компаниями, побуждает собирать все данные, которые могут быть полезны в будущем для принятия бизнес-решений, а также подбирать механизмы, чтобы своевременно эти данные анализировать. Именно для этого и нужны СУБД с OLAP.

С точки зрения бизнеса, OLAP позволяет компаниям постоянно планировать, анализировать и оценивать операционную деятельность, чтобы повышать её эффективность, уменьшать затраты и как следствие — увеличивать долю рынка. Это можно делать как в собственной системе, так и в облачной (SaaS), в веб или мобильных аналитических приложениях, CRM-системах и т.д. Технология OLAP используется во многих приложениях BI (Business Intelligence — бизнес-аналитика).

ClickHouse — это СУБД с OLAP, которая часто используется для поддержки SaaS-решений для анализа данных в различных предметных областях. Но поскольку некоторые компании все еще не слишком охотно размещают свои данные в облаке (у сторонних провайдеров), ClickHouse может быть развернут и на собственных серверах заказчика.

## OLAP с технической точки зрения

Все СУБД можно разделить на две группы: OLAP (**аналитическая** обработка в реальном времени) и OLTP (обработка **транзакций** в реальном времени). OLAP используются для построения отчетов на основе больших объемов накопленных исторических данных, но эти отчеты обновляются не слишком часто. OLTP обычно применяются для обработки непрерывных потоков операций (транзакций), каждая из которых изменяет состояние данных.

На практике OLAP и OLTP — это не строго разделённые категории, а скорее спектр возможностей. Большинство СУБД специализируются на каком-то одном виде обработки данных, но имеют инструменты и для выполнения других операций, когда это необходимо. Из-за такой специализации часто приходится использовать несколько СУБД и интегрировать их между собой. Это вполне реальная и решаемая задача, но, как известно, чем больше систем, тем выше расходы на их содержание. Поэтому в последние годы становятся популярны гибридные СУБД — НТАР (**Hybrid Transactional/Analytical Processing**), которые одинаково эффективно выполняют оба вида операций по обработке данных.

Даже если СУБД сначала развивались исключительно как OLAP или как OLTP, разработчики постепенно двигаются в сторону НТАР, чтобы сохранять конкурентоспособность. И ClickHouse не исключение. Изначально он создавался как **OLAP СУБД с максимальной производительностью**, и на сегодняшний день в нем нет полноценной поддержки обработки транзакций, но уже реализованы некоторые возможности, такие как постоянная скорость чтения/записи данных и мутации при изменении/удалении данных.

Принципиальное "разделение труда" между OLAP и OLTP СУБД сохраняется:

- Чтобы эффективно строить аналитические отчеты, нужно уметь обрабатывать колонки по отдельности, поэтому большинство OLAP СУБД — **столбцовые**.
- Хранение данных по столбцам снижает скорость выполнения операций над строками (таких как добавление или изменение данных) пропорционально числу столбцов, а это число может быть огромным для систем, ориентированных на сбор разнообразных детальных данных о событиях. Поэтому большинство OLTP систем используют строковые СУБД.

## Что такое столбцовая (колоночная) база данных?

В столбцовой БД данные каждого столбца хранятся отдельно (независимо) от других столбцов. Такой принцип хранения позволяет при выполнении запроса считывать с диска данные только тех столбцов, которые непосредственно участвуют в этом запросе. Обратная сторона такого принципа хранения заключается в том, что выполнение операций над строками становится более затратным. ClickHouse — типичный пример столбцовой СУБД.

Ключевые преимущества столбцовой СУБД:

- выполнение запросов над отдельными столбцами таблицы, а не над всей таблицей сразу;
- агрегация запросов на больших объемах данных;
- сжатие данных в столбцах.

Ниже — иллюстрация того, как извлекаются данные для отчетов при использовании обычной строковой СУБД и столбцовой СУБД:

#### **Стандартная строковая СУБД**



#### **Столбцовая СУБД**



Для аналитических приложений столбцовые СУБД предпочтительнее, так как в них можно хранить много столбцов в таблице просто на всякий случай, и это не будет сказываться на скорости чтения данных. Столбцовые СУБД предназначены для обработки и хранения больших данных. Они прекрасно масштабируются при помощи распределенных кластеров на относительно недорогих серверах — для увеличения производительности. В ClickHouse для этого используются **распределенные и реплицированные** таблицы.

## **Почему бы не использовать системы типа MapReduce?**

Системами типа MapReduce будем называть системы распределённых вычислений, в которых операция свёртки реализована на основе распределённой сортировки. Наиболее распространённое решение с открытым кодом в данном классе — [Apache Hadoop](#). Яндекс пользуется собственным решением — YT.

Такие системы не подходят для онлайн запросов в силу слишком большой задержки. То есть не могут быть использованы в качестве бэкенда для веб-интерфейса. Также эти системы не подходят для обновления данных в реальном времени. Распределённая сортировка является не оптимальным способом для выполнения операции свёртки в случае запросов, выполняющихся в режиме онлайн, потому что результат выполнения операции и все промежуточные результаты (если такие есть) помещаются в оперативную память на одном сервере. В таком случае оптимальным способом выполнения операции свёртки является хеш-таблица. Частым способом оптимизации "map-reduce" задач является предагрегация (частичная свёртка) с использованием хеш-таблицы в оперативной памяти. Пользователь делает эту оптимизацию в ручном режиме. Распределённая сортировка — основная причина тормозов при выполнении несложных задач типа "map-reduce".

Большинство реализаций MapReduce позволяют выполнять произвольный код на кластере. Но для OLAP-задач лучше подходит декларативный язык запросов, который позволяет быстро проводить исследования. Например, для Hadoop существуют Hive и Pig. Также посмотрите на Cloudera Impala, Shark (устаревший) для Spark, а также Spark SQL, Presto, Apache Drill. Впрочем, производительность

при выполнении таких задач очень неоптимальная, если сравнивать со специализированными системами, а относительно высокая задержка не позволяет использовать эти системы в качестве бэкенда для веб-интерфейса.

## Вопросы о применении ClickHouse

Вопросы:

- **Можно ли использовать ClickHouse как БД временных рядов?**
- **Можно ли использовать ClickHouse для хранения данных вида "ключ-значение"?**

### Можно ли использовать ClickHouse как базу данных временных рядов?

ClickHouse — это универсальное решение для **OLAP** операций, в то время как существует много специализированных СУБД временных рядов. Однако **высокая скорость выполнения запросов** позволяет ClickHouse во многих случаях "побеждать" специализированные аналоги. В подтверждение этому есть много примеров с конкретными показателями производительности, так что мы не будем останавливаться на этом подробно. Лучше рассмотрим те возможности ClickHouse, которые стоит использовать.

Во-первых, есть **специальные кодеки**, которые составляют типичные временные ряды. Это могут быть либо стандартные алгоритмы, такие как `DoubleDelta` или `Gorilla`, либо специфические для ClickHouse, например `T64`.

Во-вторых, запросы по временным рядам часто затрагивают только недавние данные, не старше одного дня или недели. Имеет смысл использовать серверы, где есть как быстрые диски nVME/SSD, так и более медленные, но ёмкие HDD диски. С помощью **TTL** можно сконфигурировать таблицы так, чтобы свежие данные хранились на быстрых дисках, а по мере устаревания перемещались на медленные диски. Для архивных данных можно также настроить сворачивание или даже удаление, если это необходимо.

Несмотря на то, что работа с "сырыми" данными противоречит философии ClickHouse, если нужно соответствовать очень жестким требованиям по скорости обработки данных, вы можете использовать **материализованные представления**.

### Можно ли использовать ClickHouse для хранения данных вида "ключ-значение"?

Если отвечать коротко, то "**нет**". Операции над данными вида "ключ-значение" занимают одну из верхних позиций в списке ситуаций, когда категорически **не стоит** использовать ClickHouse. Это **OLAP** СУБД, в то время как есть много специализированных СУБД для данных вида "ключ-значение".

Тем не менее, в некоторых ситуациях имеет смысл использовать ClickHouse для запросов над данными вида "ключ-значение". Чаще всего это относится к системам с относительно невысокой нагрузкой, в которых основной объем операций относится к аналитической обработке данных и отлично подходит для ClickHouse. Однако в них есть некий второстепенный процесс, в котором нужно обрабатывать данные вида "ключ-значение", при этом процесс не требует слишком высокой производительности и не имеет строгих ограничений по задержкам выполнения запросов. Если у вас нет ограничений по бюджету, вы можете использовать для таких операций вспомогательную базу данных "ключ-значение", но это увеличит расходы на обслуживание еще одной СУБД (мониторинг, бэкапы и т.д.).

Если вы все же решите не следовать рекомендациям и использовать ClickHouse для работы с данными вида "ключ-значение", вот несколько советов:

- Главная причина, по которой точечный запрос в ClickHouse становится ресурсозатратным — это разреженный индекс для первичного ключа в [таблице семейства MergeTree](#). Этот индекс не может обращаться напрямую к каждой строке данных, вместо этого он обращается к каждой N-ой строке, а затем сканирует соседние строки вплоть до указанной, обрабатывая по пути лишние данные. При обработке данных вида "ключ-значение" может быть полезно уменьшить значение N при помощи настройки `index_granularity`.
- ClickHouse хранит столбцы в отдельных файлах, поэтому чтобы собрать одну полную строку, ему приходится обрабатывать все эти файлы. Их количество растет линейно в зависимости от количества столбцов, поэтому при обработке данных вида "ключ-значение" стоит избегать использования множества столбцов и поместить все нужные данные в один столбец с типом `String` в формате JSON, Protobuf или другом подходящем формате.
- Подумайте об использовании табличного движка [Join](#) вместо обычных таблиц [MergeTree](#) и функции [joinGet](#) для получения данных. В этом случае производительность выполнения запросов может быть выше, но могут появиться проблемы с надежностью и удобством. Пример такого использования описан [здесь](#).

## Вопросы о производительности серверов и кластеров ClickHouse

Вопросы:

- [Какую версию ClickHouse использовать?](#)
- [Возможно ли удалить старые записи из таблицы ClickHouse?](#)

### Если вы не нашли то, что искали

Загляните в другие подразделы F.A.Q. или поищите в остальных разделах документации, ориентируйтесь по оглавлению слева.

[Original article](#)

## Какую версию ClickHouse использовать?

Во-первых, давайте обсудим, почему возникает этот вопрос. Есть две основные причины:

1. ClickHouse развивается достаточно быстро, и обычно мы выпускаем более 10 стабильных релизов в год. Так что есть из чего выбрать, а это не всегда просто.
2. Некоторые пользователи не хотят тратить время на анализ того, какая версия лучше подходит для их задач, и просто хотят получить совет от эксперта.

Вторая причина более весомая, так что начнем с нее, а затем рассмотрим, какие бывают релизы ClickHouse.

## Какую версию ClickHouse вы посоветуете?

Казалось бы, самый удобный вариант — нанять консультанта или довериться эксперту, и делегировать ему ответственность за вашу систему. Вы устанавливаете ту версию ClickHouse, которую вам рекомендовали, и теперь если что-то пойдет не так — это уже не ваша вина. На самом деле это не так. Никто не может знать лучше вас, что происходит в вашей системе.

Как же правильно выбрать версию ClickHouse, на которую стоит обновиться? Или как выбрать версию, с которой следует начать, если вы только внедряете ClickHouse? Во-первых, мы рекомендуем позаботиться о создании **реалистичной тестовой среды** (pre-production). В идеальном мире это была бы полная копия рабочей среды, но чаще всего такое решение оказывается слишком дорогостоящим.

Чтобы тестовая среда была достаточно надежной, но не слишком дорогостоящей, учитывайте следующие моменты:

- В тестовой среде нужно выполнять набор запросов, максимально близкий к тому, который будет выполняться в реальной среде:
  - Не используйте тестовую среду в режиме "только для чтения", работая с каким-то статичным набором данных.
  - Не используйте её в режиме "только для записи", проверяя лишь копирование данных, без построения типовых отчетов.
  - Не очищайте её, удаляя все данные подчистую вместо тестирования рабочих схем миграции.
- Выполняйте реальные запросы на выборке из реальных рабочих данных. Постарайтесь подготовить репрезентативную выборку, на которой запрос `SELECT` будет возвращать адекватные результаты. Если регламенты безопасности не позволяют использовать реальные данные за пределами защищенной рабочей среды, используйте обfuscацию.
- Убедитесь, что тестовая среда находится под контролем тех же систем мониторинга и оповещения, что и рабочая.
- Если ваша рабочая среда распределена между разными дата-центрами и регионами, тестовая среда должна быть такой же.
- Если в рабочей среде используются сложные инструменты типа репликации, распределённых таблиц или каскадных материализованных представлений, тестовая среда должна быть сконфигурирована так же.
- Обычно в тестовой среде стараются использовать то же количество серверов и виртуальных машин, что и в рабочей, но делают их меньшего объема. Либо наоборот, используют существенно меньшее число серверов и ВМ, но тех же объемов. Первый вариант скорее позволит обнаружить проблемы, связанные с работой сети, а второй вариант более прост в управлении.

Второе направление — **автоматизированное тестирование**. Не думайте, что если какой-то запрос отработал успешно один раз, так будет всегда. Считается приемлемым выполнять некоторые юнит-тесты, используя "заглушки" вместо запросов к СУБД. Но вы должны проводить достаточно большое количество автотестов, где запросы выполняются в реальном ClickHouse, чтобы убедиться, что все важные задачи отрабатывают должным образом.

В продолжение этой темы, вы можете поделиться вашими автотестами и передать их в открытую тестовую среду ClickHouse, которая используется для постоянного развития нашей СУБД. Вам придётся потратить немного времени и сил, чтобы научиться **составлять и выполнять тесты**, а также чтобы перенести ваши тесты на эту платформу. Наградой за это станет уверенность в том, что новые стабильные релизы ClickHouse будут корректно работать на ваших задачах. Это гораздо лучше, чем тратить время на то, чтобы вновь отлавливать прежние ошибки в новых версиях, а

затем ждать, пока их исправят и включат эти исправления в очередной релиз. Некоторые компании уже включили в корпоративные регламенты необходимость передачи своих тестов в ClickHouse, прежде всего стоит упомянуть [правило Beyonce](#), действующее в Google.

После того, как вы подготовили тестовую среду и инфраструктуру, выбор версии ClickHouse упрощается:

1. Проверяйте новые релизы ClickHouse с помощью подготовленных автотестов. Вы можете проверять не только стабильные релизы, но и тестовые, хотя работать с такими релизами не рекомендуется.
2. Если новый релиз ClickHouse успешно прошел ваши автотесты, внедряйте его в тестовой среде и проверяйте работоспособность всех ваших задач.
3. Сообщайте обо всех обнаруженных проблемах в [ClickHouse GitHub Issues](#).
4. Если никаких серьезных проблем не было выявлено, можно установить новый релиз ClickHouse в рабочую среду. Чтобы еще больше снизить риски, вы можете внедрить специальные техники поэтапного перехода на новые релизы, такие как [canary releases](#) или [green-blue deployments](#).

Как вы уже поняли, ClickHouse не требует какого-то особенного подхода — описанные выше правила широко используются для любых элементов инфраструктуры, если нужно обеспечить ее надежность и если компании серьезно подходят к вопросам стабильности своих систем.

## Какой вид релиза ClickHouse выбрать?

Если вы заглянете в раздел, где публикуются установочные пакеты ClickHouse, вы увидите там следующие виды пакетов:

1. `testing`
2. `prestable`
3. `stable`
4. `LTS` (long-term support)

Как уже упоминалось выше, тестовые релизы (`testing`) стоит использовать для раннего обнаружения ошибок, в рабочей среде мы не рекомендуем использовать такие релизы, поскольку они еще не протестированы так же тщательно, как остальные.

Подготовительные (`prestable`) — это релизы-кандидаты, которые с большой вероятностью скоро будут доведены до стабильного состояния. Вы можете использовать их в тестовой среде и сообщать нам об обнаруженных ошибках.

В рабочей среде мы рекомендуем использовать либо стабильный релиз (`stable`), либо релиз с долговременной поддержкой (`LTS`). Если вы выбираете между этими двумя видами релизов, примите во внимание следующее:

- По умолчанию мы рекомендуем релизы `stable`. Новый стабильный релиз выпускается примерно раз в месяц, что открывает доступ к новым функциям. Три последних стабильных релиза находятся на поддержке — это означает, что в них интегрируются исправленные ошибки и доработки.
- Релизы `LTS` выпускаются дважды в год и находятся на поддержке в течение года с момента выхода. Они более предпочтительны в следующих случаях:
  - ваши корпоративные регламенты запрещают частые обновления или использование любых релизов, кроме LTS;
  - вы используете ClickHouse в продуктах, которые не задействуют сложные инструменты ClickHouse, или у вас не хватает ресурсов для частого их обновления.

Часто компании, которые изначально ориентировались на релизы `LTS`, позднее переходят на `stable`, поскольку хотят быстрее получать доступ к новым возможностям.

## Важно

Мы всегда стремимся поддерживать совместимость релизов, но иногда это правило нарушается, и какие-то отдельные возможности в новых релизах становятся недоступны. Перед обновлением ClickHouse обязательно изучите [журнал изменений](#), чтобы убедиться, что в нем нет объявлений о нарушении обратной совместимости.

# Возможно ли удалить старые записи из таблицы ClickHouse?

Если отвечать коротко, то да. В ClickHouse есть множество механизмов, которые позволяют освобождать место на диске, удаляя старые данные. Каждый механизм подходит для разных сценариев.

## TTL

ClickHouse позволяет автоматически удалять данные при выполнении некоторых условий. Эти условия задаются как выражение, вычисляемое на основе значений любых столбцов, обычно это просто разница между текущим моментом времени и значением какого-то столбца, содержащего дату и время.

Ключевое преимущество такого подхода в том, что не нужно использовать внешнюю систему, чтобы запустить процесс — когда заданы условия TTL, удаление данных выполняется автоматически в фоновом режиме.

## Note

TTL можно использовать не только для перемещения в [/dev/null](#), но еще и между дисками, например, с SSD на HDD.

[Подробнее о конфигурировании TTL.](#)

## ALTER DELETE

ClickHouse не удаляет данные в реальном времени, как СУБД [OLTP](#). Больше всего на такое удаление похожи мутации. Они выполняются с помощью запросов `ALTER ... DELETE` или `ALTER ... UPDATE`. В отличие от обычных запросов `DELETE` и `UPDATE`, мутации выполняются асинхронно, в пакетном режиме, не в реальном времени. В остальном после слов `ALTER TABLE` синтаксис обычных запросов и мутаций одинаковый.

`ALTER DELETE` можно использовать для гибкого удаления устаревших данных. Если вам нужно делать это регулярно, единственный недостаток такого подхода будет заключаться в том, что потребуется внешняя система для запуска запроса. Кроме того, могут возникнуть некоторые проблемы с производительностью, поскольку мутации перезаписывают целые куски данных если в них содержится хотя бы одна строка, которую нужно удалить.

Это самый распространенный подход к тому, чтобы обеспечить соблюдение принципов [GDPR](#) в вашей системе на ClickHouse.

Подробнее смотрите в разделе [Мутации](#).

## DROP PARTITION

Запрос `ALTER TABLE ... DROP PARTITION` позволяет эффективно удалять целые партиции. Этот способ не такой гибкий, важно правильно сконфигурировать партиции при создании таблицы, но он подходит для достаточно широкого спектра типовых задач. Как и для мутаций, для регулярного запуска таких запросов нужна внешняя система.

Подробнее смотрите в разделе [Манипулирование с партициями и кусками](#).

## TRUNCATE

Это достаточно радикальный способ, он удаляет все данные в таблице, но хорошо подходит для отдельных случаев.

Подробнее смотрите в разделе об [удалении партиций](#).

# Интеграция ClickHouse с другими системами

Вопросы:

- [Как экспортить данные из ClickHouse в файл?](#)
- [Как импортировать JSON в ClickHouse?](#)
- [Что делать, если у меня проблема с кодировками при использовании Oracle через ODBC?](#)

## Если вы не нашли то, что искали

Загляните в другие подразделы F.A.Q. или поищите в остальных разделах документации, ориентируйтесь по оглавлению слева.

[Original article](#)

# Как экспортить данные из ClickHouse в файл?

## Секция INTO OUTFILE

Добавьте к своему запросу секцию [INTO OUTFILE](#).

Например:

```
SELECT * FROM table INTO OUTFILE 'file';
```

По умолчанию при выдаче данных ClickHouse использует формат [TabSeparated](#). Чтобы выбрать другой [формат данных](#), используйте секцию [FORMAT](#).

Например:

```
SELECT * FROM table INTO OUTFILE 'file' FORMAT CSV;
```

## Таблица с движком File

Смотрите [File](#).

# Перенаправление в командной строке

```
$ clickhouse-client --query "SELECT * from table" --format FormatName > result.txt
```

Смотрите [clickhouse-client](#).

## Как импортировать JSON в ClickHouse?

ClickHouse поддерживает широкий спектр [входных и выходных форматов данных](#). Среди них есть множество вариаций JSON, но чаще всего для импорта данных используют [JSONEachRow](#): один JSON-объект в строке, каждый объект с новой строки.

### Примеры

С помощью [HTTP-интерфейса](#):

```
$ echo '{"foo":"bar"}' | curl 'http://localhost:8123/?query=INSERT%20INTO%20test%20FORMAT%20JSONEachRow' --data-binary @-
```

При помощи [интерфейса CLI](#):

```
$ echo '{"foo":"bar"}' | clickhouse-client --query="INSERT INTO test FORMAT JSONEachRow"
```

Чтобы не вставлять данные вручную, используйте одну из [готовых библиотек](#).

### Полезные настройки

- `input_format_skip_unknown_fields` позволяет импортировать JSON, даже если он содержит дополнительные поля, которых нет в таблице (отбрасывая лишние поля).
- `input_format_import_nested_json` позволяет импортировать вложенные JSON-объекты в столбцы типа [Nested](#).

### Примечание

В HTTP-интерфейсе настройки передаются через параметры `GET` запроса, в CLI `interface` — как дополнительные аргументы командной строки, начинающиеся с `--`.

## Что делать, если у меня проблема с кодировками при использовании Oracle через ODBC?

Если вы используете Oracle через драйвер ODBC в качестве источника внешних словарей, необходимо задать правильное значение для переменной окружения `NLS_LANG` в `/etc/default/clickhouse`. Подробнее читайте в [Oracle NLS\\_LANG FAQ](#).

### Пример

```
NLS_LANG=RUSSIAN_RUSSIA.UTF8
```

# ClickHouse F.A.Q.

В этом разделе документации собраны ответы на вопросы о ClickHouse, которые задают чаще всего.

Категории:

- **Общие вопросы**

- Что такое ClickHouse?
- Почему ClickHouse такой быстрый?
- Кто пользуется ClickHouse?
- Что обозначает название ClickHouse?
- Как фраза "Не тормозит" осталась на всех футболках?
- Что такое OLAP?
- Что такое столбцовая база данных?
- Почему бы не использовать системы типа MapReduce?

- **Применение**

- Можно ли использовать ClickHouse как БД временных рядов?
- Можно ли использовать ClickHouse для хранения данных вида "ключ-значение"?

- **Операции**

- Какую версию ClickHouse использовать?
- Возможно ли удалить старые записи из таблицы ClickHouse?

- **Интеграция**

- Как экспортить данные из ClickHouse в файл?
- Как импортировать JSON в ClickHouse?
- Что делать, если у меня проблема с кодировками при использовании Oracle через 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 (Kseniia 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 (Kseniia 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 (Kseniia 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 ([Kseniia 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 (Kseniia 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). (Ksenia 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). (Ksenia 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 (Ksenia 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 (Ksenia 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 Golshtain).
- 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, got UInt8` 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_read_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 it 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 Levakov).
- 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 (Kseniia 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 (Kseniia 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 (Kseniia 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 \(overshov\)](#) ([Kseniia 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 (Kseniia 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 (sundyl).

## 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 formated 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](#) ([Kseniia 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 `YAMLParse` 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 `systemdistribution_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_read_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 (Ksenia 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 ([Kseniia 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 ([Kseniia 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](#) ([Kseniia 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 (Kseniia 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 (Kseniia 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](#) ([Ksenia 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 (changvvb).
- 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 (Izma) 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 PREWHERE 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 `Ilvm-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 `111111111.222`). 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](#) ([Kseniia 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 (mxzlxxy).
- 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](#) ([Ksenia 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 ldap 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 (Kseniia 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 whether 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 (sundyl).
- 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](#) ([sundyl](#)).
- 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 xxx;`. [#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_coldown` 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 (Ksenia 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` 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](#)).
- 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 it's 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 ( $> \text{min\_chunk\_bytes\_for\_parallel\_parsing}$ ), 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` 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 desireable. It will not skip zero values in the result. Fixes #16087. #16631 (Ildus Kurbangaliev).
- Better update of ZooKeeper configuration in runtime. #16630 (sundyli).
- 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 \(Kseniia Sumarokova\)](#).
- 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_regex_escaping_rule` 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 enumerate 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 (hcz).
- 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, FreeBSD) to clang-11. [#15622 \(Ilya Yatsishin\)](#).
- Now all test images use `llvm-symbolizer-11`. [#15069 \(alesapin\)](#).
- Allow to build with `llvm-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 or 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-lts, 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 (sundyli).
- 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-lts, 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 \(sundyli\)](#).
- 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. [#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 `lc->isNullable()` nor calling `ls->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 Golshtein](#)).

- 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 (hc).
- 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 (sundyli).
- 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 `Icm` 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. #12756 (sundyli).
- 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 (hcz).
- 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](#) ([Ksenia 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. It's 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 ZooKeeper 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.query_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 (hcz).
- 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](#) ([Darío](#)).
- 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 (Matwey 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 `Ild` 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 (Matwey 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 `PREWHERE`. Fixes #12053, #12060. #12060 (Nikolai Kochetov).
- Fixed error `Expected single dictionary argument for function` for function `defaultValueOfType` 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 downloading 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 setting is also useful with `allow_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 (hcz)
- 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 (hcz)
- 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](#))

- Now layout type for dictionaries with no arguments can be specified without round brackets in dictionaries DDL-queries. Fixes #10057. #10064 (alesapin)
- Add ability to use number ranges with leading zeros in filepath #9989 (Olga Khvostikova)
- Better memory usage in CROSS JOIN. #10029 (Artem Zuikov)
- Try to connect to all shards in cluster when getting structure of remote table and skip\_unavailable\_shards is set. #7278 (nvartolomei)
- Add `total_rows/total_bytes` into the `system.tables` table. #9919 (Azat Khuzhin)
- System log tables now use polymorphic parts by default. #9905 (Anton Popov)
- Add type column into `system.settings/merge_tree_settings` #9909 (Azat Khuzhin)
- Check for available CPU instructions at server startup as early as possible. #9888 (alexey-milovidov)
- Remove `ORDER BY` stage from mutations because we read from a single ordered part in a single thread. Also add check that the rows in mutation are ordered by sorting key and this order is not violated. #9886 (alesapin)
- Implement operator LIKE for `FixedString` at left hand side. This is needed to better support TPC-DS queries. #9890 (alexey-milovidov)
- Add `force_optimize_skip_unused_shards_no_nested` that will disable `force_optimize_skip_unused_shards` for nested Distributed table #9812 (Azat Khuzhin)
- Now columns size is calculated only once for MergeTree data parts. #9827 (alesapin)
- Evaluate constant expressions for `optimize_skip_unused_shards` (i.e. `SELECT * FROM foo_dist WHERE key=xxHash32(0)`) #8846 (Azat Khuzhin)
- Check for using `Date` or `DateTime` column from TTL expressions was removed. #9967 (Vladimir Chebotarev)
- DiskS3 hard links optimal implementation. #9760 (Pavel Kovalenko)
- If `set multiple_joins_rewriter_version = 2` enables second version of multiple JOIN rewrites that keeps not clashed column names as is. It supports multiple JOINs with `USING` and allow `select *` for JOINs with subqueries. #9739 (Artem Zuikov)
- Implementation of "non-blocking" alter for StorageMergeTree #9606 (alesapin)
- Add MergeTree full support for DiskS3 #9646 (Pavel Kovalenko)
- Extend `splitByString` to support empty strings as separators. #9742 (hczi)
- 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 flacky 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 `PRESHERE`. 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_wait_timeout_milliseconds`, which is one minute by default. The first setting is also useful with `allow_read_expired_keys` settings (to forbid reading very expired keys). #10337 ([Nikita Mikhaylov](#)).

### 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 (hc).
- 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 (hc).

## 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 (hcza)
- 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 `hdbs`. If the path contains wildcards, the table will be readonly. Example of usage: `select * from hdbs('hdbs://hdbs1:9000/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)
- Suppport 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 \(Artem Zuikov\)](#)
- 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 \(Artem Zuikov\)](#)
- 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 \(Artem Zuikov\)](#)

- 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 crtio.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 run, 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 `ALTER`s. #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+FFFDF). [#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 (hcza)
- 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 `multilf`, `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](#).

---

## Security Changelog

Исправлено в релизе 21.4.3.21, 2021-04-12

CVE-2021-25263

Злоумышленник с доступом к созданию словарей может читать файлы на файловой системе сервера Clickhouse.

Злоумышленник может обойти некорректную проверку пути к файлу словаря и загрузить часть любого файла как словарь. При этом, манипулируя опциями парсинга файла, можно получить следующую часть файла и пошагово прочитать весь файл.

Исправление доступно в версиях 20.8.18.32-lts, 21.1.9.41-stable, 21.2.9.41-stable, 21.3.6.55-lts, 21.4.3.21-stable и выше.

Обнаружено благодаря: [Вячеславу Егошину](#)

## Исправлено в релизе 19.14.3.3, 2019-09-10

### CVE-2019-15024

Злоумышленник с доступом на запись к ZooKeeper и возможностью запустить собственный сервер в сети доступной ClickHouse может создать вредоносный сервер, который будет вести себя как реплика ClickHouse и зарегистрируется в ZooKeeper. В процессе репликации вредоносный сервер может указать любой путь на файловой системе в который будут записаны данные.

Обнаружено благодаря: Эльдару Заитову из Службы Информационной Безопасности Яндекса

### CVE-2019-16535

Интерфейс декомпрессии позволял совершать ОOB чтения и записи данных в памяти, а также переполнение целочисленных переменных, что могло приводить к отказу в обслуживании. Также потенциально могло использоваться для удаленного выполнения кода.

Обнаружено благодаря: Эльдару Заитову из Службы Информационной Безопасности Яндекса

### CVE-2019-16536

Аутентифицированный клиент злоумышленника имел возможность вызвать переполнение стека, что могло привести к отказу в обслуживании.

Обнаружено благодаря: Эльдару Заитову из Службы Информационной Безопасности Яндекса

## Исправлено в релизе 19.13.6.1 от 20 сентября 2019

### CVE-2019-18657

Уязвимость в табличной функции `url` позволяла злоумышленнику добавлять произвольные HTTP-заголовки в запрос.

Обнаружено благодаря: [Никите Тихомирову](#)

## Исправлено в релизе 18.12.13 от 10 сентября 2018

### CVE-2018-14672

Функция для загрузки CatBoost моделей некорректно обрабатывала пути к файлам, что позволяло читать произвольные локальные файлы на сервере Clickhouse через сообщения об ошибках.

Обнаружено благодаря: Андрею Красичкову из Службы Информационной Безопасности Яндекса

## Исправлено в релизе 18.10.3 от 13 августа 2018

### CVE-2018-14671

unixODBC позволял указать путь для подключения произвольного shared object в качестве драйвера базы данных, что приводило к возможности выполнить произвольный код на сервере ClickHouse.

Обнаружено благодаря: Андрею Красичкову и Евгению Сидорову из Службы Информационной Безопасности Яндекса

## Исправлено в релизе 1.1.54388 от 28 июня 2018

CVE-2018-14668

Табличная функция «remote» допускала произвольные символы в полях «user», «password» и «default\_database», что позволяло производить атаки класса Cross Protocol Request Forgery.

Обнаружено благодаря: Андрею Красичкову из Службы Информационной Безопасности Яндекса

## Исправлено в релизе 1.1.54390 от 6 июля 2018

CVE-2018-14669

В ClickHouse MySQL клиенте была включена функциональность «LOAD DATA LOCAL INFILE», что позволяло получать доступ на чтение к произвольным файлам на сервере, где запущен ClickHouse.

Обнаружено благодаря: Андрею Красичкову и Евгению Сидорову из Службы Информационной Безопасности Яндекса

## Исправлено в релизе 1.1.54131 от 10 января 2017

CVE-2018-14670

Некорректная конфигурация в deb пакете могла привести к неавторизованному доступу к базе данных.

Обнаружено благодаря: the UK's National Cyber Security Centre (NCSC)

## Что нового в ClickHouse?

Планы развития вкратце изложены [здесь](#), а новости по предыдущим релизам подробно описаны в [журнале изменений](#).

Was this content helpful?

RATING\_STARS

Rating: RATING\_VALUE - RATING\_COUNT

votes