**RT.WIDESTORE**

**ИНСТРУКЦИЯ ПО УСТАНОВКЕ**

**ОГЛАВЛЕНИЕ**

[1 ВВЕДЕНИЕ 5](#_Toc75772458)

[1.1 Наименование и условное обозначение 5](#_Toc75772459)

[1.2 Область применения 5](#_Toc75772460)

[1.3 Термины, определения и сокращения 5](#_Toc75772461)

[1.4 Эксплуатационные требования 6](#_Toc75772462)

[1.4.1 Требования к Процессору 6](#_Toc75772463)

[1.4.2 Требования к оперативной памяти 6](#_Toc75772464)

[1.4.3 Файл подкачки 6](#_Toc75772465)

[1.4.4 Требования к подсистеме хранения 6](#_Toc75772466)

[1.4.5 Требование к подсистеме передачи данных 7](#_Toc75772467)

[1.4.6 Требование к операционной системе 7](#_Toc75772468)

[1.5 Требования к квалификации администратора 7](#_Toc75772469)

[2 ЭКСПЛУАТАЦИЯ RT.WIDESTORE 8](#_Toc75772470)

[2.1 Мониторинг RT.WideStore 8](#_Toc75772471)

[2.1.1 Использование аппаратных ресурсов 8](#_Toc75772472)

[2.1.2 Метрики сервера RT.WideStore 8](#_Toc75772473)

[2.2 Устранение неисправностей 9](#_Toc75772474)

[2.2.1 Соединение с сервером 9](#_Toc75772475)

[2.2.2 Сервер не запущен 9](#_Toc75772476)

[2.2.3 Параметры конфигурации 10](#_Toc75772477)

[2.2.4 Обработка запросов 10](#_Toc75772478)

[2.2.5 Скорость обработки запросов 11](#_Toc75772479)

[2.3 Управление доступом 11](#_Toc75772480)

[2.3.1 Использование 11](#_Toc75772481)

[2.3.2 Особенности реализации 12](#_Toc75772482)

[2.3.3 Аккаунт пользователя 12](#_Toc75772483)

[2.3.4 Применение настроек 13](#_Toc75772484)

[2.3.5 Роль 13](#_Toc75772485)

[2.3.6 Политика доступа к строкам 13](#_Toc75772486)

[2.3.7 Профиль настроек 14](#_Toc75772487)

[2.3.8 Квота 14](#_Toc75772488)

[2.3.9 Включение SQL-ориентированного управления доступом 14](#_Toc75772489)

[2.4 Резервное копирование данных 14](#_Toc75772490)

[2.4.1 Дублирование данных 15](#_Toc75772491)

[2.4.2 Снимки файловой системы 15](#_Toc75772492)

[2.4.3 clickhouse-copier 15](#_Toc75772493)

[2.4.4 Манипуляции с партициями 16](#_Toc75772494)

[2.5 Конфигурационные файлы 16](#_Toc75772495)

[2.6 Квоты 17](#_Toc75772496)

[2.7 Оптимизация производительности 19](#_Toc75772497)

[2.7.1 Профилировщик запросов 19](#_Toc75772498)

[2.8 Системные таблицы 20](#_Toc75772499)

[2.9 Источники системных показателей 21](#_Toc75772500)

[2.10 Настройки 22](#_Toc75772501)

[2.10.1 Пользовательские настройки 22](#_Toc75772502)

[2.10.2 Разрешения для запросов 23](#_Toc75772503)

[2.10.3 readonly 23](#_Toc75772504)

[2.10.4 allow\_ddl 23](#_Toc75772505)

[2.11 Ограничения на сложность запроса 23](#_Toc75772506)

[2.12 Другие настройки 24](#_Toc75772507)

[2.13 Профили настроек 24](#_Toc75772508)

[2.14 Ограничения на изменение настроек 25](#_Toc75772509)

[2.15 Настройки пользователей в файле user.xml 26](#_Toc75772510)

[2.15.1 user\_name/password 27](#_Toc75772511)

[2.15.2 access\_management 28](#_Toc75772512)

[2.15.3 user\_name/networks 28](#_Toc75772513)

[2.15.4 user\_name/profile 28](#_Toc75772514)

[2.15.5 user\_name/quota 28](#_Toc75772515)

[2.15.6 user\_name/databases 29](#_Toc75772516)

[ПРИЛОЖЕНИЕ 1. ПРИМЕР ИСПОЛЬЗВАНИЯ ПРОФИЛИРОВЩИКА ЗАПРОСОВ 30](#_Toc75772517)

[ПРИЛОЖЕНИЕ 2. ОПИСАНИЕ СИСТЕМНЫХ ТАБЛИЦ 44](#_Toc75772518)

[ПРИЛОЖЕНИЕ 3. ОПИСАНИЕ ПАРАМЕТРОВ ОГРАНИЧЕНИЯ НА СЛОЖНОСТЬ ЗАПРОСОВ 93](#_Toc75772519)

[ПРИЛОЖЕНИЕ 4. ОПИСАНИЕ ДОПОЛНИТЕЛЬНЫХ НАСТРОЙЕК RT.WIDESTORE 100](#_Toc75772520)

# ВВЕДЕНИЕ

## Наименование и условное обозначение

Наименование системы: RT.WideStore.

## Область применения

RT.WideStore – столбцовая система управления базами данных, предназначенная для онлайн обработки аналитических запросов (OLAP).

## Термины, определения и сокращения

В настоящем документе использованы и определены следующие термины и сокращения:

| Термин/ Сокращение | Определение |
| --- | --- |
| Docker | Платформа, которая упрощает процесс сборки, запуска, управления и распространения приложений с помощью виртуализации операционной системы, на которой он установлен. |
| OLAP | (англ. online analytical processing, интерактивная аналитическая обработка) – технология обработки данных, заключающаяся в подготовке суммарной (агрегированной) информации на основе больших массивов данных, структурированных по многомерному принципу. |
| RAM | Оперативная память. |
| RBAC | (англ. Role Based Access Contro) Управление доступом на основе ролей, развитие политики избирательного управления доступом, при этом права доступа субъектов системы на объекты группируются с учётом специфики их применения, образуя роли. |
| SQL | (англ. structured query language, язык структурированных запросов) –декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей [системой управления базами данных](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B1%D0%B0%D0%B7%D0%B0%D0%BC%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85). |
| Воркфлоу | Поток работ (англ. workflow) – графическое представление потока задач в процессе и связанных с ним подпроцессах, включая специфические работы, информационные зависимости и последовательность решений и работ. |
| ОС | Операционная система. |
| Система | Система «RT.WideStore». |
| СУБД | Система управления базами данных. |
| Шард | Сервер входящий в кластер. |

## Эксплуатационные требования

### Требования к Процессору

RT.WideStore реализует параллельную обработку данных и использует все доступные аппаратные ресурсы. При выборе процессора учитывайте, что RT.WideStore 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.

### Требования к оперативной памяти

Мы рекомендуем использовать как минимум 4 ГБ оперативной памяти, чтобы иметь возможность выполнять нетривиальные запросы. Сервер RT.WideStore 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.

RT.WideStore может использовать внешнюю память для промежуточных данных. Подробнее смотрите в разделе GROUP BY во внешней памяти.

### Файл подкачки

Отключайте файл подкачки в продуктовых средах.

### Требования к подсистеме хранения

Для установки RT.WideStore 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 RT.WideStore 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.

### Требование к подсистеме передачи данных

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.

### Требование к операционной системе

RT.WideStore разработан для семейства операционных систем Linux. Рекомендуемый дистрибутив Linux – Ubuntu. В системе должен быть установлен пакет **tzdata**.

RT.WideStore может работать и в других семействах операционных систем, например, в FreeBSD или Mac OS X.

## Требования к квалификации администратора

Администратор Системы должен иметь навыки работы со следующими продуктами:

1. Опыт администрирования ОС, на которой устанавливается Система.
2. Навыки развертывания и администрирование СУБД.
3. Опыт управление безопасностью, задание групповых политик, создание пользователей, управление удаленным доступом и т.п.
4. Знание языка запросов (SQL).
5. Навыки диагностики и мониторинга сетевого оборудования.
6. Умение работать с большим объёмом информации.
7. Владение скриптингом bash, PowerShell для оптимизации и автоматизации повседневных рутинных функций системного администрирования.
8. Опыт конфигурирования Apache Kafka, Zookeeper.
9. Опыт масштабирования СУБД.
10. Навыки технической поддержки пользователей.
11. Умение выполнять ремонт и обслуживание компьютерного и периферийного оборудования.

# ЭКСПЛУАТАЦИЯ RT.WIDESTORE

## Мониторинг RT.WideStore

Вы можете отслеживать:

* использование аппаратных ресурсов,
* метрики сервера RT.WideStore.

### Использование аппаратных ресурсов

RT.WideStore не отслеживает состояние аппаратных ресурсов самостоятельно.

Рекомендуем контролировать:

* Загрузку и температуру процессоров.

Например, можно использовать команды операционной системы: dmesg – для вывода сообщений драйверов устройств и turbostat – для вывода данных о топологии процессора, частоты, статистики состояния питания в режиме ожидания, температуре и мощности на X86 процессорах.

* Использование системы хранения, оперативной памяти и сети.

### Метрики сервера RT.WideStore

Сервер RT.WideStore имеет встроенные инструменты мониторинга.

Для отслеживания событий на сервере используйте логи. Параметры логгирования настраиваются секции [logger](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/#server_configuration_parameters-logger) конфигурационного файла.

RT.WideStore собирает:

* различные метрики того, как сервер использует вычислительные ресурсы,
* общую статистику обработки запросов.

Метрики находятся в таблицах:

* 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.).
* system.events – содержит информацию о количестве событий, произошедших в системе. Например, в таблице можно найти, сколько запросов SELECT обработано с момента запуска сервера RT.WideStore.
* system.asynchronous\_metrics – содержит метрики, которые периодически вычисляются в фоновом режиме. Например, объём используемой оперативной памяти.

Можно настроить экспорт метрик из RT.WideStore в [Graphite](https://github.com/graphite-project), для отображения в виде графика. Смотрите секцию [graphite](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/#server_configuration_parameters-graphite) конфигурационного файла RT.WideStore. Перед настройкой экспорта метрик необходимо настроить Graphite, как указано в [официальном руководстве](https://graphite.readthedocs.io/en/latest/install.html).

Можно настроить экспорт метрик из RT.WideStore в [Prometheus](https://prometheus.io/). Смотрите секцию [prometheus](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/#server_configuration_parameters-prometheus) конфигурационного файла RT.WideStore. Перед настройкой экспорта метрик необходимо настроить Prometheus, как указано в [официальном руководстве](https://prometheus.io/docs/prometheus/latest/installation/).

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](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-max_replica_delay_for_distributed_queries) и использовать HTTP ресурс /replicas\_status. Если реплика доступна и не отстаёт от других реплик, то запрос к /replicas\_status возвращает 200 OK. Если реплика отстаёт, то запрос возвращает 503 HTTP\_SERVICE\_UNAVAILABLE, включая информацию о размере отставания.

## Устранение неисправностей

Основные типы проблем, связанных с установкой и экслуатацией RT.WideStore:

* [установка дистрибутива](https://clickhouse.tech/docs/ru/operations/troubleshooting/#troubleshooting-installation-errors),
* [соединение с сервером](https://clickhouse.tech/docs/ru/operations/troubleshooting/#troubleshooting-accepts-no-connections),
* [обработка запросов](https://clickhouse.tech/docs/ru/operations/troubleshooting/#troubleshooting-does-not-process-queries),
* [скорость обработки запросов](https://clickhouse.tech/docs/ru/operations/troubleshooting/#troubleshooting-too-slow).

### Соединение с сервером

Возможные проблемы соединения с сервером:

* сервер не запущен,
* неожиданные или неправильные параметры конфигурации.

### Сервер не запущен

Если сервер не запущен:

**Проверьте, запущен ли сервер командой:**

$ 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](server\_configuration\_parameters/settings.md#server\_configuration\_parameters-listen\_host)

и

[tcp\_port](server\_configuration\_parameters/settings.md#server\_configuration\_parameters-tcp\_port).

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 RT.WideStore не может обработать запрос, он отправляет клиенту описание ошибки. В clickhouse-client вы получаете описание ошибки в консоли. При использовании интерфейса HTTP, RT.WideStore 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 c параметром stack-trace, то RT.WideStore 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 RT.WideStore 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.

## Управление доступом

RT.WideStore поддерживает управление доступом на основе подхода RBAC, то есть основанную на управление доступом на основе ролей.

Объекты системы доступа в RT.WideStore:

* [аккаунт пользователя](https://clickhouse.tech/docs/ru/operations/access-rights/#user-account-management),
* [роль](https://clickhouse.tech/docs/ru/operations/access-rights/#role-management),
* [политика доступа к строкам](https://clickhouse.tech/docs/ru/operations/access-rights/#row-policy-management),
* [профиль настроек](https://clickhouse.tech/docs/ru/operations/access-rights/#settings-profiles-management),
* [квота](https://clickhouse.tech/docs/ru/operations/access-rights/#quotas-management).

Вы можете настроить объекты системы доступа, используя:

* SQL-ориентированный воркфлоу.

Функциональность необходимо [включить](https://clickhouse.tech/docs/ru/operations/access-rights/#enabling-access-control) управление доступом (см. п.2.3.9).

* [Конфигурационные файлы](https://clickhouse.tech/docs/ru/operations/configuration-files/) 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](https://clickhouse.tech/docs/ru/sql-reference/statements/show/#show-access-statement).

### Использование

По умолчанию сервер RT.WideStore 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](https://clickhouse.tech/docs/ru/engines/table-engines/special/distributed/).

Если вы начали пользоваться RT.WideStore недавно, попробуйте следующий сценарий:

1. [Включите](https://clickhouse.tech/docs/ru/operations/access-rights/#enabling-access-control) 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](https://clickhouse.tech/docs/ru/operations/settings/permissions-for-queries/#permissions_for_queries) для пользователя default и отключите для него SQL-ориентированное управление доступом.

### Особенности реализации

Существуют следующие особенности реализации выдачи разрешений:

* 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 RT.WideStore. Аккаунт содержит:

* Идентификационную информацию.
* [Привилегии](https://clickhouse.tech/docs/ru/sql-reference/statements/grant/#grant-privileges), определяющие область действия запросов, которые могут быть выполнены пользователем.
* Хосты, которые могут подключаться к серверу RT.WideStore.
* 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](https://clickhouse.tech/docs/ru/sql-reference/statements/grant/) или через назначение [ролей](https://clickhouse.tech/docs/ru/operations/access-rights/#role-management). Отозвать привилегию можно с помощью запроса [REVOKE](https://clickhouse.tech/docs/ru/sql-reference/statements/revoke/). Чтобы вывести список присвоенных привилегий, используется выражение [SHOW GRANTS](https://clickhouse.tech/docs/ru/sql-reference/statements/show/#show-grants-statement).

Запросы управления:

* [CREATE USER](https://clickhouse.tech/docs/ru/sql-reference/statements/create/user/#create-user-statement),
* [ALTER USER](https://clickhouse.tech/docs/ru/sql-reference/statements/alter/user/),
* [DROP USER](https://clickhouse.tech/docs/ru/sql-reference/statements/misc/#drop-user-statement),
* [SHOW CREATE USER](https://clickhouse.tech/docs/ru/sql-reference/statements/show/#show-create-user-statement).

### 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](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/#default-profile) (профили настроек default\_profile находятся в файле, указанном в параметре user\_config см. Конфигурационные параметры сервера).

### Роль

Роль – это контейнер объектов системы доступа, которые можно присвоить аккаунту пользователя.

Роль содержит:

1. [Привилегии](https://clickhouse.tech/docs/ru/sql-reference/statements/grant/#grant-privileges) (привелегия – разрешение на выполнение определенного типа запросов, она имеет иерархическую структуру. Набор разрешенных запросов зависит от области действия привилегии).
2. Настройки и ограничения.
3. Список назначенных ролей.
4. Запросы управления:

* [CREATE ROLE](https://clickhouse.tech/docs/ru/sql-reference/statements/create/#create-role-statement),
* [ALTER ROLE](https://clickhouse.tech/docs/ru/sql-reference/statements/alter/role/),
* [DROP ROLE](https://clickhouse.tech/docs/ru/sql-reference/statements/misc/#drop-role-statement),
* [SET ROLE](https://clickhouse.tech/docs/ru/sql-reference/statements/misc/#set-role-statement),
* [SET DEFAULT ROLE](https://clickhouse.tech/docs/ru/sql-reference/statements/misc/#set-default-role-statement),
* [SHOW CREATE ROLE](https://clickhouse.tech/docs/ru/sql-reference/statements/show/#show-create-role-statement).

Привилегии можно присвоить роли с помощью запроса [GRANT](https://clickhouse.tech/docs/ru/sql-reference/statements/grant/). Для отзыва привилегий у роли RT.WideStore предоставляет запрос [REVOKE](https://clickhouse.tech/docs/ru/sql-reference/statements/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](https://clickhouse.tech/docs/ru/sql-reference/statements/create/#create-row-policy-statement),
* [ALTER ROW POLICY](https://clickhouse.tech/docs/ru/sql-reference/statements/alter/row-policy/),
* [DROP ROW POLICY](https://clickhouse.tech/docs/ru/sql-reference/statements/misc/#drop-row-policy-statement),
* [SHOW CREATE ROW POLICY](https://clickhouse.tech/docs/ru/sql-reference/statements/show/#show-create-row-policy-statement).

### Профиль настроек

Профиль настроек – это набор [настроек](https://clickhouse.tech/docs/ru/operations/settings/). 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](https://clickhouse.tech/docs/ru/sql-reference/statements/create/#create-settings-profile-statement),
* [ALTER SETTINGS PROFILE](https://clickhouse.tech/docs/ru/sql-reference/statements/alter/settings-profile/),
* [DROP SETTINGS PROFILE](https://clickhouse.tech/docs/ru/sql-reference/statements/misc/#drop-settings-profile-statement),
* [SHOW CREATE SETTINGS PROFILE](https://clickhouse.tech/docs/ru/sql-reference/statements/show/#show-create-settings-profile-statement).

### Квота

Квота ограничивает использование ресурсов. См. п. [2.6](https://clickhouse.tech/docs/ru/operations/quotas/).

Квота содержит набор ограничений определенной длительности, а также список ролей и/или пользователей, на которых распространяется данная квота.

Запросы управления:

* [CREATE QUOTA](https://clickhouse.tech/docs/ru/sql-reference/statements/create/#create-quota-statement),
* [ALTER QUOTA](https://clickhouse.tech/docs/ru/sql-reference/statements/alter/quota/),
* [DROP QUOTA](https://clickhouse.tech/docs/ru/sql-reference/statements/misc/#drop-quota-statement),
* [SHOW CREATE QUOTA](https://clickhouse.tech/docs/ru/sql-reference/statements/show/#show-create-quota-statement).

### Включение SQL-ориентированного управления доступом

Для включения SQL-ориентированного управления доступом:

* Настройте каталог для хранения конфигураций.

RT.WideStore хранит конфигурации объектов системы доступа в каталоге, установленном в конфигурационном параметре сервера [access\_control\_path](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/#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](https://clickhouse.tech/docs/ru/operations/settings/settings-users/#access_management-user-setting).

## Резервное копирование данных

[Репликация](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/replication/) 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. RT.WideStore имеет встроенные средства защиты для предотвращения некоторых типов ошибок – например, по умолчанию [не получится удалить таблицы \*MergeTree, содержащие более 50 Гб данных, одной командой](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/#max-table-size-to-drop). 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 RT.WideStore, 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 RT.WideStore.*

### Дублирование данных

Часто данные, которые поступают в RT.WideStore, доставляются через некоторую отказоустойчивую очередь, например [Apache Kafka](https://kafka.apache.org/). В этом случае можно настроить дополнительный набор подписчиков, которые будут считывать один и тот же поток данных во время записи в RT.WideStore 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](https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html).

### Снимки файловой системы

Некоторые локальные файловые системы позволяют делать снимки (например, [ZFS](https://en.wikipedia.org/wiki/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](https://clickhouse.tech/docs/ru/engines/table-engines/special/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](https://clickhouse.tech/docs/ru/operations/utilities/clickhouse-copier/) – это универсальный инструмент, который изначально был создан для перешардирования таблиц с петабайтами данных. Его также можно использовать для резервного копирования и восстановления, поскольку он надёжно копирует данные между таблицами и кластерами RT.WideStore.

Для небольших объёмов данных можно применять "INSERT INTO ... SELECT ..." в удалённые таблицы.

### Манипуляции с партициями

RT.WideStore позволяет использовать запрос "ALTER TABLE ... FREEZE PARTITION ..." для создания локальной копии партиций 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 RT.WideStore, 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](https://en.wikipedia.org/wiki/Rsync)).

Для автоматизации этого подхода доступен инструмент от сторонних разработчиков: [clickhouse-backup](https://github.com/AlexAkulov/clickhouse-backup) (инструмент для простого резервного копирования и восстановления RT.WideStore с поддержкой облачных хранилищ).

## Конфигурационные файлы

Основной конфигурационный файл сервера – config.xml. Он расположен в директории /etc/clickhouse-server/.

Отдельные настройки могут быть переопределены в файлах \*.xml и \*.conf из директории config.d рядом с конфигом.

У элементов этих конфигурационных файлов могут быть указаны атрибуты replace или remove.

Если ни один не указан – объединить содержимое элементов рекурсивно с заменых значений совпадающих детей.

Если указано replace – заменить весь элемент на указанный.

Если указано remove – удалить элемент.

Также в конфиге 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](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/#server_configuration_parameters-include_from). Значения подстановок указываются в элементах /widestore/имя\_подстановки этого файла. Если подстановка, заданная в incl отсутствует, то в лог попадает соответствующая запись. Чтобы RT.WideStore не писал в лог об отсутствии подстановки, необходимо указать атрибут optional="true" (например, настройка [macros](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/)).

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.

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

<widestore>

<users>

<alice>

<profile>analytics</profile>

<networks>

<ip>::/0</ip>

</networks>

<password\_sha256\_hex>...</password\_sha256\_hex>

<quota>analytics</quota>

</alice>

</users>

</widestore>

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.

## 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 этого смотрите 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>

<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>

<errors>100</errors>

<result\_rows>1000000000</result\_rows>

<read\_rows>100000000000</read\_rows>

<execution\_time>900</execution\_time>

</interval>

<interval>

<duration>86400</duration>

<queries>10000</queries>

<errors>1000</errors>

<result\_rows>5000000000</result\_rows>

<read\_rows>500000000000</read\_rows>

<execution\_time>7200</execution\_time>

</interval>

</statbox>

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;
* errors – количество запросов, при выполнении которых было выкинуто исключение;
* result\_rows – суммарное количество строк, отданных 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.

## Оптимизация производительности

### Профилировщик запросов

RT.WideStore запускает профилировщик выборки, позволяющий анализировать выполнение запроса. Используя профилировщик, вы можете найти процедуры исходного кода, которые наиболее часто используются во время выполнения запроса. Вы можете отслеживать время процессора и реальное время, включая время простоя.

Чтобы использовать профилировщик:

* Настройте раздел [trace\_log](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/#server_configuration_parameters-trace_log) конфигурации сервера.

В этом разделе настраивается системная таблица [trace\_log,](https://clickhouse.tech/docs/ru/operations/system-tables/trace_log/#system_tables-trace_log) содержащая результаты работы профилировщика. Он настроен по умолчанию. Помните, что данные в этой таблице действительны только для работающего сервера. После перезапуска сервера RT.WideStore не очищает таблицу, и весь сохраненный адрес виртуальной памяти может стать недействительным.

* Установите параметры [query\_profiler\_cpu\_time\_period\_ns](https://clickhouse.tech/docs/ru/operations/settings/settings/#query_profiler_cpu_time_period_ns) или [query\_profiler\_real\_time\_period\_ns](https://clickhouse.tech/docs/ru/operations/settings/settings/#query_profiler_real_time_period_ns). Обе настройки можно использовать одновременно.

Эти параметры позволяют настраивать таймеры профилировщика. Поскольку это настройки сеанса, вы можете получить различную частоту выборки для всего сервера, отдельных пользователей или профилей пользователей, для вашего интерактивного сеанса и для каждого отдельного запроса.

Частота дискретизации по умолчанию – одна выборка в секунду, при этом включены как ЦП, так и таймеры реального времени. Такая частота позволяет собрать достаточно информации о кластере RT.WideStore. При этом, работая с такой периодичностью, профилировщик не влияет на производительность сервера RT.WideStore. Если вам нужно профилировать каждый отдельный запрос, попробуйте использовать более высокую частоту выборки.

Чтобы проанализировать системную таблицу trace\_log:

* Установите clickhouse-common-static-dbg пакет.
* Разрешите функции самоанализа с [помощью](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-allow_introspection_functions) параметра [allow\_introspection\_functions](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-allow_introspection_functions).
* По соображениям безопасности функции самоанализа по умолчанию отключены.
* Используйте [функции самоанализа а](https://clickhouse.tech/docs/ru/sql-reference/functions/introspection/)addressToLine, addressToSymbolи, чтобы получить имена функций и их позиции в коде RT.WideStore. Чтобы получить профиль для какого-либо запроса, вам нужно агрегировать данные из таблицы trace\_log. Вы можете агрегировать данные по отдельным функциям или по всем трассировкам стека.

Если вам нужно визуализировать trace\_log информацию, используйте [flamegraph](https://clickhouse.tech/docs/ru/operations/interfaces/third-party/gui/#clickhouse-flamegraph) и [speedscope](https://github.com/laplab/clickhouse-speedscope).

Пример использования профилировщика запросов представлен в Приложении 1.

## Системные таблицы

Системные таблицы содержат следующую информацию:

* состоянии сервера, процессов и окружении,
* внутренние процессы сервера.

Системные таблицы:

* находятся в базе данных system,
* доступны только для чтения,
* не могут быть удалены или изменены, но их можно отключить.

Большинство системных таблиц хранят свои данные в оперативной памяти. Сервер RT.WideStore создает эти системные таблицы при старте.

В отличие от других системных таблиц, таблицы с системными логами [metric\_log](https://clickhouse.tech/docs/ru/operations/system-tables/metric_log/), [query\_log](https://clickhouse.tech/docs/ru/operations/system-tables/query_log/), [query\_thread\_log](https://clickhouse.tech/docs/ru/operations/system-tables/query_thread_log/), [trace\_log](https://clickhouse.tech/docs/ru/operations/system-tables/trace_log/), [part\_log](https://clickhouse.tech/docs/ru/operations/system-tables/part_log/), [crash\_log](https://clickhouse.tech/docs/ru/operations/system-tables/crash-log/) и [text\_log](https://clickhouse.tech/docs/ru/operations/system-tables/text_log/) используют движок таблиц [MergeTree](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/) и по умолчанию хранят свои данные в файловой системе. Если удалить таблицу из файловой системы, сервер RT.WideStore снова создаст пустую таблицу во время следующей записи данных. Если схема системной таблицы изменилась в новом релизе, то RT.WideStore 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](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/custom-partitioning-key/).
* ttl – [время жизни](https://clickhouse.tech/docs/ru/sql-reference/statements/alter/ttl/) таблицы.
* 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:

<widestore>

<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>

</widestore>

По умолчанию размер таблицы не ограничен. Управлять размером таблицы можно используя [TTL](https://clickhouse.tech/docs/ru/sql-reference/statements/alter/ttl/#manipuliatsii-s-ttl-tablitsy) для удаления устаревших записей журнала. Также вы можете использовать функцию партиционирования для таблиц MergeTree.

Описание системных таблиц представлено в Приложении 2.

## Источники системных показателей

Для сбора системных показателей сервер RT.WideStore использует:

* возможности CAP\_NET\_ADMIN,
* [procfs](https://ru.wikipedia.org/wiki/Procfs) (только Linux).

Если для сервера RT.WideStore 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 RT.WideStore собирает следующие системные показатели:

* OSCPUVirtualTimeMicroseconds,
* OSCPUWaitMicroseconds,
* OSIOWaitMicroseconds,
* OSReadChars,
* OSWriteChars,
* OSReadBytes,
* OSWriteBytes.

## 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 RT.WideStore в интерактивном режиме отправьте запрос SET setting=value.
* Аналогично можно использовать RT.WideStore-сессии в HTTP-протоколе, для этого необходимо указывать HTTP-параметр session\_id.
* Настройки для запроса:
  + При запуске консольного клиента RT.WideStore 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](https://clickhouse.tech/docs/ru/sql-reference/statements/select/#settings-in-select) 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](https://clickhouse.tech/docs/ru/operations/settings/settings/), 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](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/#custom_settings_prefixes) конфигурационнного 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');

### Разрешения для запросов

Запросы в RT.WideStore можно разделить на несколько типов:

* Запросы на чтение данных: SELECT, SHOW, DESCRIBE, EXISTS.
* Запросы за запись данных: INSERT, OPTIMIZE.
* Запросы на изменение настроек: SET, USE.
* [Запросы DDL](https://ru.wikipedia.org/wiki/Data_Definition_Language): CREATE, ALTER, RENAME, ATTACH, DETACH, DROP TRUNCATE.
* KILL QUERY.

Разрешения пользователя по типу запроса регулируются параметрами:

* [readonly](https://clickhouse.tech/docs/ru/operations/settings/permissions-for-queries/#settings_readonly) – ограничивает разрешения для всех типов запросов, кроме DDL.
* [allow\_ddl](https://clickhouse.tech/docs/ru/operations/settings/permissions-for-queries/#settings_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](https://clickhouse.tech/docs/ru/operations/settings/permissions-for-queries/#permissions_for_queries) 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](https://clickhouse.tech/docs/ru/interfaces/http/#http-interface), 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](https://clickhouse.tech/docs/ru/operations/settings/constraints-on-settings/).

Значение по умолчанию: 0.

### allow\_ddl

Разрешает/запрещает [DDL](https://ru.wikipedia.org/wiki/Data_Definition_Language) запросы.

Разделение запросов по типам смотрите по тексту [выше](https://clickhouse.tech/docs/ru/operations/settings/permissions-for-queries/#permissions_for_queries) 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.

Параметры ограничения на сложность запросов представлены в *Приложении 3*.

## Другие настройки

Описание дополнительных настройек RT.WideStore представлены в *Приложении 4*.

## Профили настроек

Профиль настроек – это набор настроек, сгруппированных под одним именем.

***Примечание:*** *Для управления профилями настроек рекомендуется использовать* [*SQL-ориентированный воркфлоу*](https://clickhouse.tech/docs/ru/operations/access-rights/#access-control)*, который также поддерживается в RT.WideStore.*

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 quries from the user interface -->

<web>

<max\_rows\_to\_read>1000000000</max\_rows\_to\_read>

<max\_bytes\_to\_read>100000000000</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>1000000000</max\_bytes\_to\_sort>

<max\_result\_rows>100000</max\_result\_rows>

<max\_result\_bytes>100000000</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 обязателен 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 в файле user.xml

Раздел users конфигурационного файла user.xml содержит настройки для пользователей.

***Примечание:*** *Для управления пользователями рекомендуется использовать* [*SQL-ориентированный воркфлоу*](https://clickhouse.tech/docs/ru/operations/access-rights/#access-control)*, который также поддерживается в RT.WideStore.*

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>

<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" | sha1sum | tr -d '-' | xxd -r -p | sha1sum | tr -d '-'

```

Первая строка результата — пароль. Вторая строка — соответствующий ему двойной хэш SHA1.

### access\_management

Включает или выключает SQL-ориентированное [управление доступом](https://clickhouse.tech/docs/ru/operations/access-rights/#access-control) для пользователя.

Возможные значения:

* 0 – выключено.
* 1 – включено.

Значение по умолчанию: 0.

### user\_name/networks

Список сетей, из которых пользователь может подключиться к серверу RT.WideStore.

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 п. 2.13.

### 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 п. 2.6.

### user\_name/databases

В этом разделе вы можете ограничить выдачу RT.WideStore 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 содержать любое выражение, возвращающее значение типа [UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/). 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.

ПРИЛОЖЕНИЕ 1. ПРИМЕР ИСПОЛЬЗВАНИЯ ПРОФИЛИРОВЩИКА ЗАПРОСОВ

В этом примере мы:

* Фильтруем в trace\_log данные по идентификатору запроса и текущей дате.
* Агрегируем данные по трассировке стека.
* Используем функции интроспекции, для получения отчета:
  + Названия и соответствующие функции запроса.
  + Расположение исходного кода этих функций.

SELECT

count(),

arrayStringConcat(arrayMap(x -> concat(demangle(addressToSymbol(x)), '\n ', addressToLine(x)), trace), '\n') AS sym

FROM system.trace\_log

WHERE (query\_id = 'ebca3574-ad0a-400a-9cbc-dca382f5998c') AND (event\_date = today())

GROUP BY trace

ORDER BY count() DESC

LIMIT 10

Row 1:

──────

count(): 6344

sym: StackTrace::StackTrace(ucontext\_t const&)

/home/milovidov/ClickHouse/build\_gcc9/../src/Common/StackTrace.cpp:208

DB::(anonymous namespace)::writeTraceInfo(DB::TimerType, int, siginfo\_t\*, void\*) [clone .isra.0]

/home/milovidov/ClickHouse/build\_gcc9/../src/IO/BufferBase.h:99

read

DB::ReadBufferFromFileDescriptor::nextImpl()

/home/milovidov/ClickHouse/build\_gcc9/../src/IO/ReadBufferFromFileDescriptor.cpp:56

DB::CompressedReadBufferBase::readCompressedData(unsigned long&, unsigned long&)

/home/milovidov/ClickHouse/build\_gcc9/../src/IO/ReadBuffer.h:54

DB::CompressedReadBufferFromFile::nextImpl()

/home/milovidov/ClickHouse/build\_gcc9/../src/Compression/CompressedReadBufferFromFile.cpp:22

DB::CompressedReadBufferFromFile::seek(unsigned long, unsigned long)

/home/milovidov/ClickHouse/build\_gcc9/../src/Compression/CompressedReadBufferFromFile.cpp:63

DB::MergeTreeReaderStream::seekToMark(unsigned long)

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeReaderStream.cpp:200

std::\_Function\_handler<DB::ReadBuffer\* (std::vector<DB::IDataType::Substream, std::allocator<DB::IDataType::Substream> > const&), DB::MergeTreeReader::readData(std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> > const&, DB::IDataType const&, DB::IColumn&, unsigned long, bool, unsigned long, bool)::{lambda(bool)#1}::operator()(bool) const::{lambda(std::vector<DB::IDataType::Substream, std::allocator<DB::IDataType::Substream> > const&)#1}>::\_M\_invoke(std::\_Any\_data const&, std::vector<DB::IDataType::Substream, std::allocator<DB::IDataType::Substream> > const&)

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeReader.cpp:212

DB::IDataType::deserializeBinaryBulkWithMultipleStreams(DB::IColumn&, unsigned long, DB::IDataType::DeserializeBinaryBulkSettings&, std::shared\_ptr<DB::IDataType::DeserializeBinaryBulkState>&) const

/usr/local/include/c++/9.1.0/bits/std\_function.h:690

DB::MergeTreeReader::readData(std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> > const&, DB::IDataType const&, DB::IColumn&, unsigned long, bool, unsigned long, bool)

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeReader.cpp:232

DB::MergeTreeReader::readRows(unsigned long, bool, unsigned long, DB::Block&)

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeReader.cpp:111

DB::MergeTreeRangeReader::DelayedStream::finalize(DB::Block&)

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeRangeReader.cpp:35

DB::MergeTreeRangeReader::continueReadingChain(DB::MergeTreeRangeReader::ReadResult&)

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeRangeReader.cpp:219

DB::MergeTreeRangeReader::read(unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange> >&)

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeRangeReader.cpp:487

DB::MergeTreeBaseSelectBlockInputStream::readFromPartImpl()

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeBaseSelectBlockInputStream.cpp:158

DB::MergeTreeBaseSelectBlockInputStream::readImpl()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::IBlockInputStream::read()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::ExpressionBlockInputStream::readImpl()

/home/milovidov/ClickHouse/build\_gcc9/../src/DataStreams/ExpressionBlockInputStream.cpp:34

DB::IBlockInputStream::read()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::PartialSortingBlockInputStream::readImpl()

/home/milovidov/ClickHouse/build\_gcc9/../src/DataStreams/PartialSortingBlockInputStream.cpp:13

DB::IBlockInputStream::read()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::loop(unsigned long)

/usr/local/include/c++/9.1.0/bits/atomic\_base.h:419

DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::thread(std::shared\_ptr<DB::ThreadGroupStatus>, unsigned long)

/home/milovidov/ClickHouse/build\_gcc9/../src/DataStreams/ParallelInputsProcessor.h:215

ThreadFromGlobalPool::ThreadFromGlobalPool<void (DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::\*)(std::shared\_ptr<DB::ThreadGroupStatus>, unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>\*, std::shared\_ptr<DB::ThreadGroupStatus>, unsigned long&>(void (DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::\*&&)(std::shared\_ptr<DB::ThreadGroupStatus>, unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>\*&&, std::shared\_ptr<DB::ThreadGroupStatus>&&, unsigned long&)::{lambda()#1}::operator()() const

/usr/local/include/c++/9.1.0/bits/shared\_ptr\_base.h:729

ThreadPoolImpl<std::thread>::worker(std::\_List\_iterator<std::thread>)

/usr/local/include/c++/9.1.0/bits/unique\_lock.h:69

execute\_native\_thread\_routine

/home/milovidov/ClickHouse/ci/workspace/gcc/gcc-build/x86\_64-pc-linux-gnu/libstdc++-v3/include/bits/unique\_ptr.h:81

start\_thread

\_\_clone

Row 2:

──────

count(): 3295

sym: StackTrace::StackTrace(ucontext\_t const&)

/home/milovidov/ClickHouse/build\_gcc9/../src/Common/StackTrace.cpp:208

DB::(anonymous namespace)::writeTraceInfo(DB::TimerType, int, siginfo\_t\*, void\*) [clone .isra.0]

/home/milovidov/ClickHouse/build\_gcc9/../src/IO/BufferBase.h:99

\_\_pthread\_cond\_wait

std::condition\_variable::wait(std::unique\_lock<std::mutex>&)

/home/milovidov/ClickHouse/ci/workspace/gcc/gcc-build/x86\_64-pc-linux-gnu/libstdc++-v3/src/c++11/../../../../../gcc-9.1.0/libstdc++-v3/src/c++11/condition\_variable.cc:55

Poco::Semaphore::wait()

/home/milovidov/ClickHouse/build\_gcc9/../contrib/poco/Foundation/src/Semaphore.cpp:61

DB::UnionBlockInputStream::readImpl()

/usr/local/include/c++/9.1.0/x86\_64-pc-linux-gnu/bits/gthr-default.h:748

DB::IBlockInputStream::read()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::MergeSortingBlockInputStream::readImpl()

/home/milovidov/ClickHouse/build\_gcc9/../src/Core/Block.h:90

DB::IBlockInputStream::read()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::ExpressionBlockInputStream::readImpl()

/home/milovidov/ClickHouse/build\_gcc9/../src/DataStreams/ExpressionBlockInputStream.cpp:34

DB::IBlockInputStream::read()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::LimitBlockInputStream::readImpl()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::IBlockInputStream::read()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::AsynchronousBlockInputStream::calculate()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

std::\_Function\_handler<void (), DB::AsynchronousBlockInputStream::next()::{lambda()#1}>::\_M\_invoke(std::\_Any\_data const&)

/usr/local/include/c++/9.1.0/bits/atomic\_base.h:551

ThreadPoolImpl<ThreadFromGlobalPool>::worker(std::\_List\_iterator<ThreadFromGlobalPool>)

/usr/local/include/c++/9.1.0/x86\_64-pc-linux-gnu/bits/gthr-default.h:748

ThreadFromGlobalPool::ThreadFromGlobalPool<ThreadPoolImpl<ThreadFromGlobalPool>::scheduleImpl<void>(std::function<void ()>, int, std::optional<unsigned long>)::{lambda()#3}>(ThreadPoolImpl<ThreadFromGlobalPool>::scheduleImpl<void>(std::function<void ()>, int, std::optional<unsigned long>)::{lambda()#3}&&)::{lambda()#1}::operator()() const

/home/milovidov/ClickHouse/build\_gcc9/../src/Common/ThreadPool.h:146

ThreadPoolImpl<std::thread>::worker(std::\_List\_iterator<std::thread>)

/usr/local/include/c++/9.1.0/bits/unique\_lock.h:69

execute\_native\_thread\_routine

/home/milovidov/ClickHouse/ci/workspace/gcc/gcc-build/x86\_64-pc-linux-gnu/libstdc++-v3/include/bits/unique\_ptr.h:81

start\_thread

\_\_clone

Row 3:

──────

count(): 1978

sym: StackTrace::StackTrace(ucontext\_t const&)

/home/milovidov/ClickHouse/build\_gcc9/../src/Common/StackTrace.cpp:208

DB::(anonymous namespace)::writeTraceInfo(DB::TimerType, int, siginfo\_t\*, void\*) [clone .isra.0]

/home/milovidov/ClickHouse/build\_gcc9/../src/IO/BufferBase.h:99

DB::VolnitskyBase<true, true, DB::StringSearcher<true, true> >::search(unsigned char const\*, unsigned long) const

/opt/milovidov/ClickHouse/build\_gcc9/programs/clickhouse

DB::MatchImpl<true, false>::vector\_constant(DB::PODArray<unsigned char, 4096ul, AllocatorWithHint<false, AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul> const&, DB::PODArray<unsigned long, 4096ul, AllocatorWithHint<false, AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul> const&, std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> > const&, DB::PODArray<unsigned char, 4096ul, AllocatorWithHint<false, AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul>&)

/opt/milovidov/ClickHouse/build\_gcc9/programs/clickhouse

DB::FunctionsStringSearch<DB::MatchImpl<true, false>, DB::NameLike>::executeImpl(DB::Block&, std::vector<unsigned long, std::allocator<unsigned long> > const&, unsigned long, unsigned long)

/opt/milovidov/ClickHouse/build\_gcc9/programs/clickhouse

DB::PreparedFunctionImpl::execute(DB::Block&, std::vector<unsigned long, std::allocator<unsigned long> > const&, unsigned long, unsigned long, bool)

/home/milovidov/ClickHouse/build\_gcc9/../src/Functions/IFunction.cpp:464

DB::ExpressionAction::execute(DB::Block&, bool) const

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:677

DB::ExpressionActions::execute(DB::Block&, bool) const

/home/milovidov/ClickHouse/build\_gcc9/../src/Interpreters/ExpressionActions.cpp:739

DB::MergeTreeRangeReader::executePrewhereActionsAndFilterColumns(DB::MergeTreeRangeReader::ReadResult&)

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeRangeReader.cpp:660

DB::MergeTreeRangeReader::read(unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange> >&)

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeRangeReader.cpp:546

DB::MergeTreeRangeReader::read(unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange> >&)

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::MergeTreeBaseSelectBlockInputStream::readFromPartImpl()

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeBaseSelectBlockInputStream.cpp:158

DB::MergeTreeBaseSelectBlockInputStream::readImpl()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::IBlockInputStream::read()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::ExpressionBlockInputStream::readImpl()

/home/milovidov/ClickHouse/build\_gcc9/../src/DataStreams/ExpressionBlockInputStream.cpp:34

DB::IBlockInputStream::read()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::PartialSortingBlockInputStream::readImpl()

/home/milovidov/ClickHouse/build\_gcc9/../src/DataStreams/PartialSortingBlockInputStream.cpp:13

DB::IBlockInputStream::read()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::loop(unsigned long)

/usr/local/include/c++/9.1.0/bits/atomic\_base.h:419

DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::thread(std::shared\_ptr<DB::ThreadGroupStatus>, unsigned long)

/home/milovidov/ClickHouse/build\_gcc9/../src/DataStreams/ParallelInputsProcessor.h:215

ThreadFromGlobalPool::ThreadFromGlobalPool<void (DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::\*)(std::shared\_ptr<DB::ThreadGroupStatus>, unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>\*, std::shared\_ptr<DB::ThreadGroupStatus>, unsigned long&>(void (DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::\*&&)(std::shared\_ptr<DB::ThreadGroupStatus>, unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>\*&&, std::shared\_ptr<DB::ThreadGroupStatus>&&, unsigned long&)::{lambda()#1}::operator()() const

/usr/local/include/c++/9.1.0/bits/shared\_ptr\_base.h:729

ThreadPoolImpl<std::thread>::worker(std::\_List\_iterator<std::thread>)

/usr/local/include/c++/9.1.0/bits/unique\_lock.h:69

execute\_native\_thread\_routine

/home/milovidov/ClickHouse/ci/workspace/gcc/gcc-build/x86\_64-pc-linux-gnu/libstdc++-v3/include/bits/unique\_ptr.h:81

start\_thread

\_\_clone

Row 4:

──────

count(): 1913

sym: StackTrace::StackTrace(ucontext\_t const&)

/home/milovidov/ClickHouse/build\_gcc9/../src/Common/StackTrace.cpp:208

DB::(anonymous namespace)::writeTraceInfo(DB::TimerType, int, siginfo\_t\*, void\*) [clone .isra.0]

/home/milovidov/ClickHouse/build\_gcc9/../src/IO/BufferBase.h:99

DB::VolnitskyBase<true, true, DB::StringSearcher<true, true> >::search(unsigned char const\*, unsigned long) const

/opt/milovidov/ClickHouse/build\_gcc9/programs/clickhouse

DB::MatchImpl<true, false>::vector\_constant(DB::PODArray<unsigned char, 4096ul, AllocatorWithHint<false, AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul> const&, DB::PODArray<unsigned long, 4096ul, AllocatorWithHint<false, AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul> const&, std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> > const&, DB::PODArray<unsigned char, 4096ul, AllocatorWithHint<false, AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul>&)

/opt/milovidov/ClickHouse/build\_gcc9/programs/clickhouse

DB::FunctionsStringSearch<DB::MatchImpl<true, false>, DB::NameLike>::executeImpl(DB::Block&, std::vector<unsigned long, std::allocator<unsigned long> > const&, unsigned long, unsigned long)

/opt/milovidov/ClickHouse/build\_gcc9/programs/clickhouse

DB::PreparedFunctionImpl::execute(DB::Block&, std::vector<unsigned long, std::allocator<unsigned long> > const&, unsigned long, unsigned long, bool)

/home/milovidov/ClickHouse/build\_gcc9/../src/Functions/IFunction.cpp:464

DB::ExpressionAction::execute(DB::Block&, bool) const

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:677

DB::ExpressionActions::execute(DB::Block&, bool) const

/home/milovidov/ClickHouse/build\_gcc9/../src/Interpreters/ExpressionActions.cpp:739

DB::MergeTreeRangeReader::executePrewhereActionsAndFilterColumns(DB::MergeTreeRangeReader::ReadResult&)

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeRangeReader.cpp:660

DB::MergeTreeRangeReader::read(unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange> >&)

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeRangeReader.cpp:546

DB::MergeTreeRangeReader::read(unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange> >&)

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::MergeTreeBaseSelectBlockInputStream::readFromPartImpl()

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeBaseSelectBlockInputStream.cpp:158

DB::MergeTreeBaseSelectBlockInputStream::readImpl()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::IBlockInputStream::read()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::ExpressionBlockInputStream::readImpl()

/home/milovidov/ClickHouse/build\_gcc9/../src/DataStreams/ExpressionBlockInputStream.cpp:34

DB::IBlockInputStream::read()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::PartialSortingBlockInputStream::readImpl()

/home/milovidov/ClickHouse/build\_gcc9/../src/DataStreams/PartialSortingBlockInputStream.cpp:13

DB::IBlockInputStream::read()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::loop(unsigned long)

/usr/local/include/c++/9.1.0/bits/atomic\_base.h:419

DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::thread(std::shared\_ptr<DB::ThreadGroupStatus>, unsigned long)

/home/milovidov/ClickHouse/build\_gcc9/../src/DataStreams/ParallelInputsProcessor.h:215

ThreadFromGlobalPool::ThreadFromGlobalPool<void (DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::\*)(std::shared\_ptr<DB::ThreadGroupStatus>, unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>\*, std::shared\_ptr<DB::ThreadGroupStatus>, unsigned long&>(void (DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::\*&&)(std::shared\_ptr<DB::ThreadGroupStatus>, unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>\*&&, std::shared\_ptr<DB::ThreadGroupStatus>&&, unsigned long&)::{lambda()#1}::operator()() const

/usr/local/include/c++/9.1.0/bits/shared\_ptr\_base.h:729

ThreadPoolImpl<std::thread>::worker(std::\_List\_iterator<std::thread>)

/usr/local/include/c++/9.1.0/bits/unique\_lock.h:69

execute\_native\_thread\_routine

/home/milovidov/ClickHouse/ci/workspace/gcc/gcc-build/x86\_64-pc-linux-gnu/libstdc++-v3/include/bits/unique\_ptr.h:81

start\_thread

\_\_clone

Row 5:

──────

count(): 1672

sym: StackTrace::StackTrace(ucontext\_t const&)

/home/milovidov/ClickHouse/build\_gcc9/../src/Common/StackTrace.cpp:208

DB::(anonymous namespace)::writeTraceInfo(DB::TimerType, int, siginfo\_t\*, void\*) [clone .isra.0]

/home/milovidov/ClickHouse/build\_gcc9/../src/IO/BufferBase.h:99

DB::VolnitskyBase<true, true, DB::StringSearcher<true, true> >::search(unsigned char const\*, unsigned long) const

/opt/milovidov/ClickHouse/build\_gcc9/programs/clickhouse

DB::MatchImpl<true, false>::vector\_constant(DB::PODArray<unsigned char, 4096ul, AllocatorWithHint<false, AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul> const&, DB::PODArray<unsigned long, 4096ul, AllocatorWithHint<false, AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul> const&, std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> > const&, DB::PODArray<unsigned char, 4096ul, AllocatorWithHint<false, AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul>&)

/opt/milovidov/ClickHouse/build\_gcc9/programs/clickhouse

DB::FunctionsStringSearch<DB::MatchImpl<true, false>, DB::NameLike>::executeImpl(DB::Block&, std::vector<unsigned long, std::allocator<unsigned long> > const&, unsigned long, unsigned long)

/opt/milovidov/ClickHouse/build\_gcc9/programs/clickhouse

DB::PreparedFunctionImpl::execute(DB::Block&, std::vector<unsigned long, std::allocator<unsigned long> > const&, unsigned long, unsigned long, bool)

/home/milovidov/ClickHouse/build\_gcc9/../src/Functions/IFunction.cpp:464

DB::ExpressionAction::execute(DB::Block&, bool) const

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:677

DB::ExpressionActions::execute(DB::Block&, bool) const

/home/milovidov/ClickHouse/build\_gcc9/../src/Interpreters/ExpressionActions.cpp:739

DB::MergeTreeRangeReader::executePrewhereActionsAndFilterColumns(DB::MergeTreeRangeReader::ReadResult&)

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeRangeReader.cpp:660

DB::MergeTreeRangeReader::read(unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange> >&)

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeRangeReader.cpp:546

DB::MergeTreeRangeReader::read(unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange> >&)

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::MergeTreeBaseSelectBlockInputStream::readFromPartImpl()

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeBaseSelectBlockInputStream.cpp:158

DB::MergeTreeBaseSelectBlockInputStream::readImpl()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::IBlockInputStream::read()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::ExpressionBlockInputStream::readImpl()

/home/milovidov/ClickHouse/build\_gcc9/../src/DataStreams/ExpressionBlockInputStream.cpp:34

DB::IBlockInputStream::read()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::PartialSortingBlockInputStream::readImpl()

/home/milovidov/ClickHouse/build\_gcc9/../src/DataStreams/PartialSortingBlockInputStream.cpp:13

DB::IBlockInputStream::read()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::loop(unsigned long)

/usr/local/include/c++/9.1.0/bits/atomic\_base.h:419

DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::thread(std::shared\_ptr<DB::ThreadGroupStatus>, unsigned long)

/home/milovidov/ClickHouse/build\_gcc9/../src/DataStreams/ParallelInputsProcessor.h:215

ThreadFromGlobalPool::ThreadFromGlobalPool<void (DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::\*)(std::shared\_ptr<DB::ThreadGroupStatus>, unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>\*, std::shared\_ptr<DB::ThreadGroupStatus>, unsigned long&>(void (DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::\*&&)(std::shared\_ptr<DB::ThreadGroupStatus>, unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>\*&&, std::shared\_ptr<DB::ThreadGroupStatus>&&, unsigned long&)::{lambda()#1}::operator()() const

/usr/local/include/c++/9.1.0/bits/shared\_ptr\_base.h:729

ThreadPoolImpl<std::thread>::worker(std::\_List\_iterator<std::thread>)

/usr/local/include/c++/9.1.0/bits/unique\_lock.h:69

execute\_native\_thread\_routine

/home/milovidov/ClickHouse/ci/workspace/gcc/gcc-build/x86\_64-pc-linux-gnu/libstdc++-v3/include/bits/unique\_ptr.h:81

start\_thread

\_\_clone

Row 6:

──────

count(): 1531

sym: StackTrace::StackTrace(ucontext\_t const&)

/home/milovidov/ClickHouse/build\_gcc9/../src/Common/StackTrace.cpp:208

DB::(anonymous namespace)::writeTraceInfo(DB::TimerType, int, siginfo\_t\*, void\*) [clone .isra.0]

/home/milovidov/ClickHouse/build\_gcc9/../src/IO/BufferBase.h:99

read

DB::ReadBufferFromFileDescriptor::nextImpl()

/home/milovidov/ClickHouse/build\_gcc9/../src/IO/ReadBufferFromFileDescriptor.cpp:56

DB::CompressedReadBufferBase::readCompressedData(unsigned long&, unsigned long&)

/home/milovidov/ClickHouse/build\_gcc9/../src/IO/ReadBuffer.h:54

DB::CompressedReadBufferFromFile::nextImpl()

/home/milovidov/ClickHouse/build\_gcc9/../src/Compression/CompressedReadBufferFromFile.cpp:22

void DB::deserializeBinarySSE2<4>(DB::PODArray<unsigned char, 4096ul, AllocatorWithHint<false, AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul>&, DB::PODArray<unsigned long, 4096ul, AllocatorWithHint<false, AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul>&, DB::ReadBuffer&, unsigned long)

/home/milovidov/ClickHouse/build\_gcc9/../src/IO/ReadBuffer.h:53

DB::DataTypeString::deserializeBinaryBulk(DB::IColumn&, DB::ReadBuffer&, unsigned long, double) const

/home/milovidov/ClickHouse/build\_gcc9/../src/DataTypes/DataTypeString.cpp:202

DB::MergeTreeReader::readData(std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> > const&, DB::IDataType const&, DB::IColumn&, unsigned long, bool, unsigned long, bool)

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeReader.cpp:232

DB::MergeTreeReader::readRows(unsigned long, bool, unsigned long, DB::Block&)

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeReader.cpp:111

DB::MergeTreeRangeReader::DelayedStream::finalize(DB::Block&)

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeRangeReader.cpp:35

DB::MergeTreeRangeReader::startReadingChain(unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange> >&)

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeRangeReader.cpp:219

DB::MergeTreeRangeReader::read(unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange> >&)

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::MergeTreeRangeReader::read(unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange> >&)

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::MergeTreeBaseSelectBlockInputStream::readFromPartImpl()

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeBaseSelectBlockInputStream.cpp:158

DB::MergeTreeBaseSelectBlockInputStream::readImpl()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::IBlockInputStream::read()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::ExpressionBlockInputStream::readImpl()

/home/milovidov/ClickHouse/build\_gcc9/../src/DataStreams/ExpressionBlockInputStream.cpp:34

DB::IBlockInputStream::read()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::PartialSortingBlockInputStream::readImpl()

/home/milovidov/ClickHouse/build\_gcc9/../src/DataStreams/PartialSortingBlockInputStream.cpp:13

DB::IBlockInputStream::read()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::loop(unsigned long)

/usr/local/include/c++/9.1.0/bits/atomic\_base.h:419

DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::thread(std::shared\_ptr<DB::ThreadGroupStatus>, unsigned long)

/home/milovidov/ClickHouse/build\_gcc9/../src/DataStreams/ParallelInputsProcessor.h:215

ThreadFromGlobalPool::ThreadFromGlobalPool<void (DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::\*)(std::shared\_ptr<DB::ThreadGroupStatus>, unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>\*, std::shared\_ptr<DB::ThreadGroupStatus>, unsigned long&>(void (DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::\*&&)(std::shared\_ptr<DB::ThreadGroupStatus>, unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>\*&&, std::shared\_ptr<DB::ThreadGroupStatus>&&, unsigned long&)::{lambda()#1}::operator()() const

/usr/local/include/c++/9.1.0/bits/shared\_ptr\_base.h:729

ThreadPoolImpl<std::thread>::worker(std::\_List\_iterator<std::thread>)

/usr/local/include/c++/9.1.0/bits/unique\_lock.h:69

execute\_native\_thread\_routine

/home/milovidov/ClickHouse/ci/workspace/gcc/gcc-build/x86\_64-pc-linux-gnu/libstdc++-v3/include/bits/unique\_ptr.h:81

start\_thread

\_\_clone

Row 7:

──────

count(): 1034

sym: StackTrace::StackTrace(ucontext\_t const&)

/home/milovidov/ClickHouse/build\_gcc9/../src/Common/StackTrace.cpp:208

DB::(anonymous namespace)::writeTraceInfo(DB::TimerType, int, siginfo\_t\*, void\*) [clone .isra.0]

/home/milovidov/ClickHouse/build\_gcc9/../src/IO/BufferBase.h:99

DB::VolnitskyBase<true, true, DB::StringSearcher<true, true> >::search(unsigned char const\*, unsigned long) const

/opt/milovidov/ClickHouse/build\_gcc9/programs/clickhouse

DB::MatchImpl<true, false>::vector\_constant(DB::PODArray<unsigned char, 4096ul, AllocatorWithHint<false, AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul> const&, DB::PODArray<unsigned long, 4096ul, AllocatorWithHint<false, AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul> const&, std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> > const&, DB::PODArray<unsigned char, 4096ul, AllocatorWithHint<false, AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul>&)

/opt/milovidov/ClickHouse/build\_gcc9/programs/clickhouse

DB::FunctionsStringSearch<DB::MatchImpl<true, false>, DB::NameLike>::executeImpl(DB::Block&, std::vector<unsigned long, std::allocator<unsigned long> > const&, unsigned long, unsigned long)

/opt/milovidov/ClickHouse/build\_gcc9/programs/clickhouse

DB::PreparedFunctionImpl::execute(DB::Block&, std::vector<unsigned long, std::allocator<unsigned long> > const&, unsigned long, unsigned long, bool)

/home/milovidov/ClickHouse/build\_gcc9/../src/Functions/IFunction.cpp:464

DB::ExpressionAction::execute(DB::Block&, bool) const

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:677

DB::ExpressionActions::execute(DB::Block&, bool) const

/home/milovidov/ClickHouse/build\_gcc9/../src/Interpreters/ExpressionActions.cpp:739

DB::MergeTreeRangeReader::executePrewhereActionsAndFilterColumns(DB::MergeTreeRangeReader::ReadResult&)

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeRangeReader.cpp:660

DB::MergeTreeRangeReader::read(unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange> >&)

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeRangeReader.cpp:546

DB::MergeTreeRangeReader::read(unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange> >&)

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::MergeTreeBaseSelectBlockInputStream::readFromPartImpl()

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeBaseSelectBlockInputStream.cpp:158

DB::MergeTreeBaseSelectBlockInputStream::readImpl()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::IBlockInputStream::read()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::ExpressionBlockInputStream::readImpl()

/home/milovidov/ClickHouse/build\_gcc9/../src/DataStreams/ExpressionBlockInputStream.cpp:34

DB::IBlockInputStream::read()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::PartialSortingBlockInputStream::readImpl()

/home/milovidov/ClickHouse/build\_gcc9/../src/DataStreams/PartialSortingBlockInputStream.cpp:13

DB::IBlockInputStream::read()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::loop(unsigned long)

/usr/local/include/c++/9.1.0/bits/atomic\_base.h:419

DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::thread(std::shared\_ptr<DB::ThreadGroupStatus>, unsigned long)

/home/milovidov/ClickHouse/build\_gcc9/../src/DataStreams/ParallelInputsProcessor.h:215

ThreadFromGlobalPool::ThreadFromGlobalPool<void (DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::\*)(std::shared\_ptr<DB::ThreadGroupStatus>, unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>\*, std::shared\_ptr<DB::ThreadGroupStatus>, unsigned long&>(void (DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::\*&&)(std::shared\_ptr<DB::ThreadGroupStatus>, unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>\*&&, std::shared\_ptr<DB::ThreadGroupStatus>&&, unsigned long&)::{lambda()#1}::operator()() const

/usr/local/include/c++/9.1.0/bits/shared\_ptr\_base.h:729

ThreadPoolImpl<std::thread>::worker(std::\_List\_iterator<std::thread>)

/usr/local/include/c++/9.1.0/bits/unique\_lock.h:69

execute\_native\_thread\_routine

/home/milovidov/ClickHouse/ci/workspace/gcc/gcc-build/x86\_64-pc-linux-gnu/libstdc++-v3/include/bits/unique\_ptr.h:81

start\_thread

\_\_clone

Row 8:

──────

count(): 989

sym: StackTrace::StackTrace(ucontext\_t const&)

/home/milovidov/ClickHouse/build\_gcc9/../src/Common/StackTrace.cpp:208

DB::(anonymous namespace)::writeTraceInfo(DB::TimerType, int, siginfo\_t\*, void\*) [clone .isra.0]

/home/milovidov/ClickHouse/build\_gcc9/../src/IO/BufferBase.h:99

\_\_lll\_lock\_wait

pthread\_mutex\_lock

DB::MergeTreeReaderStream::loadMarks()

/usr/local/include/c++/9.1.0/bits/std\_mutex.h:103

DB::MergeTreeReaderStream::MergeTreeReaderStream(std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> > const&, std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> > const&, unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange> > const&, DB::MarkCache\*, bool, DB::UncompressedCache\*, unsigned long, unsigned long, unsigned long, DB::MergeTreeIndexGranularityInfo const\*, std::function<void (DB::ReadBufferFromFileBase::ProfileInfo)> const&, int)

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeReaderStream.cpp:107

std::\_Function\_handler<void (std::vector<DB::IDataType::Substream, std::allocator<DB::IDataType::Substream> > const&), DB::MergeTreeReader::addStreams(std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> > const&, DB::IDataType const&, std::function<void (DB::ReadBufferFromFileBase::ProfileInfo)> const&, int)::{lambda(std::vector<DB::IDataType::Substream, std::allocator<DB::IDataType::Substream> > const&)#1}>::\_M\_invoke(std::\_Any\_data const&, std::vector<DB::IDataType::Substream, std::allocator<DB::IDataType::Substream> > const&)

/usr/local/include/c++/9.1.0/bits/unique\_ptr.h:147

DB::MergeTreeReader::addStreams(std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> > const&, DB::IDataType const&, std::function<void (DB::ReadBufferFromFileBase::ProfileInfo)> const&, int)

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:677

DB::MergeTreeReader::MergeTreeReader(std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> > const&, std::shared\_ptr<DB::MergeTreeDataPart const> const&, DB::NamesAndTypesList const&, DB::UncompressedCache\*, DB::MarkCache\*, bool, DB::MergeTreeData const&, std::vector<DB::MarkRange, std::allocator<DB::MarkRange> > const&, unsigned long, unsigned long, std::map<std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> >, double, std::less<std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> > >, std::allocator<std::pair<std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> > const, double> > > const&, std::function<void (DB::ReadBufferFromFileBase::ProfileInfo)> const&, int)

/usr/local/include/c++/9.1.0/bits/stl\_list.h:303

DB::MergeTreeThreadSelectBlockInputStream::getNewTask()

/usr/local/include/c++/9.1.0/bits/std\_function.h:259

DB::MergeTreeBaseSelectBlockInputStream::readImpl()

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeBaseSelectBlockInputStream.cpp:54

DB::IBlockInputStream::read()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::ExpressionBlockInputStream::readImpl()

/home/milovidov/ClickHouse/build\_gcc9/../src/DataStreams/ExpressionBlockInputStream.cpp:34

DB::IBlockInputStream::read()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::PartialSortingBlockInputStream::readImpl()

/home/milovidov/ClickHouse/build\_gcc9/../src/DataStreams/PartialSortingBlockInputStream.cpp:13

DB::IBlockInputStream::read()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::loop(unsigned long)

/usr/local/include/c++/9.1.0/bits/atomic\_base.h:419

DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::thread(std::shared\_ptr<DB::ThreadGroupStatus>, unsigned long)

/home/milovidov/ClickHouse/build\_gcc9/../src/DataStreams/ParallelInputsProcessor.h:215

ThreadFromGlobalPool::ThreadFromGlobalPool<void (DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::\*)(std::shared\_ptr<DB::ThreadGroupStatus>, unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>\*, std::shared\_ptr<DB::ThreadGroupStatus>, unsigned long&>(void (DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::\*&&)(std::shared\_ptr<DB::ThreadGroupStatus>, unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>\*&&, std::shared\_ptr<DB::ThreadGroupStatus>&&, unsigned long&)::{lambda()#1}::operator()() const

/usr/local/include/c++/9.1.0/bits/shared\_ptr\_base.h:729

ThreadPoolImpl<std::thread>::worker(std::\_List\_iterator<std::thread>)

/usr/local/include/c++/9.1.0/bits/unique\_lock.h:69

execute\_native\_thread\_routine

/home/milovidov/ClickHouse/ci/workspace/gcc/gcc-build/x86\_64-pc-linux-gnu/libstdc++-v3/include/bits/unique\_ptr.h:81

start\_thread

\_\_clone

Row 9:

───────

count(): 779

sym: StackTrace::StackTrace(ucontext\_t const&)

/home/milovidov/ClickHouse/build\_gcc9/../src/Common/StackTrace.cpp:208

DB::(anonymous namespace)::writeTraceInfo(DB::TimerType, int, siginfo\_t\*, void\*) [clone .isra.0]

/home/milovidov/ClickHouse/build\_gcc9/../src/IO/BufferBase.h:99

void DB::deserializeBinarySSE2<4>(DB::PODArray<unsigned char, 4096ul, AllocatorWithHint<false, AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul>&, DB::PODArray<unsigned long, 4096ul, AllocatorWithHint<false, AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul>&, DB::ReadBuffer&, unsigned long)

/usr/local/lib/gcc/x86\_64-pc-linux-gnu/9.1.0/include/emmintrin.h:727

DB::DataTypeString::deserializeBinaryBulk(DB::IColumn&, DB::ReadBuffer&, unsigned long, double) const

/home/milovidov/ClickHouse/build\_gcc9/../src/DataTypes/DataTypeString.cpp:202

DB::MergeTreeReader::readData(std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> > const&, DB::IDataType const&, DB::IColumn&, unsigned long, bool, unsigned long, bool)

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeReader.cpp:232

DB::MergeTreeReader::readRows(unsigned long, bool, unsigned long, DB::Block&)

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeReader.cpp:111

DB::MergeTreeRangeReader::DelayedStream::finalize(DB::Block&)

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeRangeReader.cpp:35

DB::MergeTreeRangeReader::startReadingChain(unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange> >&)

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeRangeReader.cpp:219

DB::MergeTreeRangeReader::read(unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange> >&)

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::MergeTreeRangeReader::read(unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange> >&)

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::MergeTreeBaseSelectBlockInputStream::readFromPartImpl()

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeBaseSelectBlockInputStream.cpp:158

DB::MergeTreeBaseSelectBlockInputStream::readImpl()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::IBlockInputStream::read()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::ExpressionBlockInputStream::readImpl()

/home/milovidov/ClickHouse/build\_gcc9/../src/DataStreams/ExpressionBlockInputStream.cpp:34

DB::IBlockInputStream::read()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::PartialSortingBlockInputStream::readImpl()

/home/milovidov/ClickHouse/build\_gcc9/../src/DataStreams/PartialSortingBlockInputStream.cpp:13

DB::IBlockInputStream::read()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::loop(unsigned long)

/usr/local/include/c++/9.1.0/bits/atomic\_base.h:419

DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::thread(std::shared\_ptr<DB::ThreadGroupStatus>, unsigned long)

/home/milovidov/ClickHouse/build\_gcc9/../src/DataStreams/ParallelInputsProcessor.h:215

ThreadFromGlobalPool::ThreadFromGlobalPool<void (DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::\*)(std::shared\_ptr<DB::ThreadGroupStatus>, unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>\*, std::shared\_ptr<DB::ThreadGroupStatus>, unsigned long&>(void (DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::\*&&)(std::shared\_ptr<DB::ThreadGroupStatus>, unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>\*&&, std::shared\_ptr<DB::ThreadGroupStatus>&&, unsigned long&)::{lambda()#1}::operator()() const

/usr/local/include/c++/9.1.0/bits/shared\_ptr\_base.h:729

ThreadPoolImpl<std::thread>::worker(std::\_List\_iterator<std::thread>)

/usr/local/include/c++/9.1.0/bits/unique\_lock.h:69

execute\_native\_thread\_routine

/home/milovidov/ClickHouse/ci/workspace/gcc/gcc-build/x86\_64-pc-linux-gnu/libstdc++-v3/include/bits/unique\_ptr.h:81

start\_thread

\_\_clone

Row 10:

───────

count(): 666

sym: StackTrace::StackTrace(ucontext\_t const&)

/home/milovidov/ClickHouse/build\_gcc9/../src/Common/StackTrace.cpp:208

DB::(anonymous namespace)::writeTraceInfo(DB::TimerType, int, siginfo\_t\*, void\*) [clone .isra.0]

/home/milovidov/ClickHouse/build\_gcc9/../src/IO/BufferBase.h:99

void DB::deserializeBinarySSE2<4>(DB::PODArray<unsigned char, 4096ul, AllocatorWithHint<false, AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul>&, DB::PODArray<unsigned long, 4096ul, AllocatorWithHint<false, AllocatorHints::DefaultHint, 67108864ul>, 15ul, 16ul>&, DB::ReadBuffer&, unsigned long)

/usr/local/lib/gcc/x86\_64-pc-linux-gnu/9.1.0/include/emmintrin.h:727

DB::DataTypeString::deserializeBinaryBulk(DB::IColumn&, DB::ReadBuffer&, unsigned long, double) const

/home/milovidov/ClickHouse/build\_gcc9/../src/DataTypes/DataTypeString.cpp:202

DB::MergeTreeReader::readData(std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> > const&, DB::IDataType const&, DB::IColumn&, unsigned long, bool, unsigned long, bool)

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeReader.cpp:232

DB::MergeTreeReader::readRows(unsigned long, bool, unsigned long, DB::Block&)

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeReader.cpp:111

DB::MergeTreeRangeReader::DelayedStream::finalize(DB::Block&)

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeRangeReader.cpp:35

DB::MergeTreeRangeReader::startReadingChain(unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange> >&)

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeRangeReader.cpp:219

DB::MergeTreeRangeReader::read(unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange> >&)

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::MergeTreeRangeReader::read(unsigned long, std::vector<DB::MarkRange, std::allocator<DB::MarkRange> >&)

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::MergeTreeBaseSelectBlockInputStream::readFromPartImpl()

/home/milovidov/ClickHouse/build\_gcc9/../src/Storages/MergeTree/MergeTreeBaseSelectBlockInputStream.cpp:158

DB::MergeTreeBaseSelectBlockInputStream::readImpl()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::IBlockInputStream::read()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::ExpressionBlockInputStream::readImpl()

/home/milovidov/ClickHouse/build\_gcc9/../src/DataStreams/ExpressionBlockInputStream.cpp:34

DB::IBlockInputStream::read()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::PartialSortingBlockInputStream::readImpl()

/home/milovidov/ClickHouse/build\_gcc9/../src/DataStreams/PartialSortingBlockInputStream.cpp:13

DB::IBlockInputStream::read()

/usr/local/include/c++/9.1.0/bits/stl\_vector.h:108

DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::loop(unsigned long)

/usr/local/include/c++/9.1.0/bits/atomic\_base.h:419

DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::thread(std::shared\_ptr<DB::ThreadGroupStatus>, unsigned long)

/home/milovidov/ClickHouse/build\_gcc9/../src/DataStreams/ParallelInputsProcessor.h:215

ThreadFromGlobalPool::ThreadFromGlobalPool<void (DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::\*)(std::shared\_ptr<DB::ThreadGroupStatus>, unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>\*, std::shared\_ptr<DB::ThreadGroupStatus>, unsigned long&>(void (DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>::\*&&)(std::shared\_ptr<DB::ThreadGroupStatus>, unsigned long), DB::ParallelInputsProcessor<DB::UnionBlockInputStream::Handler>\*&&, std::shared\_ptr<DB::ThreadGroupStatus>&&, unsigned long&)::{lambda()#1}::operator()() const

/usr/local/include/c++/9.1.0/bits/shared\_ptr\_base.h:729

ThreadPoolImpl<std::thread>::worker(std::\_List\_iterator<std::thread>)

/usr/local/include/c++/9.1.0/bits/unique\_lock.h:69

execute\_native\_thread\_routine

/home/milovidov/ClickHouse/ci/workspace/gcc/gcc-build/x86\_64-pc-linux-gnu/libstdc++-v3/include/bits/unique\_ptr.h:81

start\_thread

\_\_clone

ПРИЛОЖЕНИЕ 2. ОПИСАНИЕ СИСТЕМНЫХ ТАБЛИЦ

##### 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](https://clickhouse.tech/docs/ru/sql-reference/data-types/date/)) – дата события.
* event\_time ([DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/)) – время события.
* event\_time\_microseconds ([DateTime64](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime64/)) – время события в микросекундах.
* name ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – название метрики.
* value ([Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/)) – 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](https://clickhouse.tech/docs/ru/operations/system-tables/asynchronous_metric_log/#system_tables-asynchronous_metrics) – содержит метрики, которые периодически вычисляются в фоновом режиме.
* [system.metric\_log](https://clickhouse.tech/docs/ru/operations/system-tables/asynchronous_metric_log/#system_tables-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](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – название метрики.
* value ([Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/)) – 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](https://clickhouse.tech/docs/ru/operations/monitoring/) – основы мониторинга в RT.WideStore.
* [system.metrics](https://clickhouse.tech/docs/ru/operations/system-tables/asynchronous_metrics/#system_tables-metrics) – таблица с мгновенно вычисляемыми метриками.
* [system.events](https://clickhouse.tech/docs/ru/operations/system-tables/asynchronous_metrics/#system_tables-events) – таблица с количеством произошедших событий.
* [system.metric\_log](https://clickhouse.tech/docs/ru/operations/system-tables/asynchronous_metrics/#system_tables-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 входят.

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) – TIP-адрес хоста, полученный из DNS.
* port (UInt16) – порт, на который обращаться для соединения с сервером.
* user (String) – имя пользователя, которого использовать для соединения с сервером.

##### system.columns

Содержит информацию о столбцах всех таблиц.

С помощью этой таблицы можно получить информацию аналогично запросу [DESCRIBE TABLE](https://clickhouse.tech/docs/ru/sql-reference/statements/misc/#misc-describe-table), но для многих таблиц сразу.

Таблица system.columns содержит столбцы (тип столбца указан в скобках):

* database ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя базы данных.
* table ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя таблицы.
* name ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя столбца.
* type ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – тип столбца.
* position ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – порядковый номер столбца в таблице (нумерация начинается с 1).
* default\_kind ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – тип выражения (DEFAULT, MATERIALIZED, ALIAS) для значения по умолчанию или пустая строка.
* default\_expression ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – выражение для значения по умолчанию или пустая строка.
* data\_compressed\_bytes ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – размер сжатых данных в байтах.
* data\_uncompressed\_bytes ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – размер распакованных данных в байтах.
* marks\_bytes ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – размер засечек в байтах.
* comment ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – комментарий к столбцу или пустая строка.
* is\_in\_partition\_key ([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – флаг, показывающий включение столбца в ключ партиционирования.
* is\_in\_sorting\_key ([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – флаг, показывающий включение столбца в ключ сортировки.
* is\_in\_primary\_key ([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – флаг, показывающий включение столбца в первичный ключ.
* is\_in\_sampling\_key ([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – флаг, показывающий включение столбца в ключ выборки.
* compression\_codec ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя кодека сжатия.

##### 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](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/)) – дата события.
* event\_time ([Datetime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/)) – время события.
* timestamp\_ns ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – время события с наносекундами.
* signal ([Int32](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – номер сигнала, пришедшего в поток.
* thread\_id ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – идентификатор треда.
* query\_id ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – идентификатор запроса.
* trace ([Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/))) – трассировка стека в момент ошибки. Представляет собой список физических адресов, по которым расположены вызываемые методы.
* trace\_full ([Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – трассировка стека в момент ошибки. Содержит вызываемые методы.
* version ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – версия сервера RT.WideStore.
* revision ([UInt32](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – ревизия сборки сервера RT.WideStore.
* build\_id ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – BuildID, сгенерированный компилятором.

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::executeImpl(std::\_\_1::vector<DB::ColumnWithTypeAndName, std::\_\_1::allocator<DB::ColumnWithTypeAndName> >&, 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](https://clickhouse.tech/docs/ru/operations/system-tables/trace_log/).

##### system.current\_roles

Содержит активные роли текущего пользователя. SET ROLE изменяет содержимое этой таблицы.

Столбцы:

* role\_name ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя роли.
* with\_admin\_option ([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – флаг, который показывает, обладает ли current\_role роль привилегией ADMIN OPTION.
* is\_default ([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – флаг, который показывает, является ли current\_role ролью по умолчанию.

##### system.data\_type\_families

Содержит информацию о поддерживаемых [типах данных](https://clickhouse.tech/docs/ru/operations/sql-reference/data-types/).

Столбцы:

* name ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя типа данных.
* case\_insensitive ([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – свойство, которое показывает, зависит ли имя типа данных в запросе от регистра. Например, допустимы и Date, и date.
* alias\_to ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/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 │

└────────────┴──────────────────┴──────────┘

Смотр также:

* [Синтаксис](https://clickhouse.tech/docs/ru/sql-reference/syntax/) – 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](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/). Столбец reason содержит причину, по которой кусок был отсоединён. Для кусов, отсоединённых пользователем, reason содержит пустую строку.

Такие куски могут быть присоединены с помощью [ALTER TABLE ATTACH PARTITION|PART](https://clickhouse.tech/docs/ru/operations/sql_reference/alter/#alter_attach-partition). Остальные столбцы описаны в [system.parts](https://clickhouse.tech/docs/ru/operations/system-tables/detached_parts/#system_tables-parts).

Если имя куска некорректно, значения некоторых столбцов могут быть NULL. Такие куски могут быть удалены с помощью [ALTER TABLE DROP DETACHED PART](https://clickhouse.tech/docs/ru/operations/sql_reference/alter/#alter_drop-detached).

##### system.dictionaries

Содержит информацию о [внешних словарях](https://clickhouse.tech/docs/ru/sql-reference/dictionaries/external-dictionaries/external-dicts/).

Столбцы:

* database ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя базы данных, в которой находится словарь, созданный с помощью DDL-запроса. Пустая строка для других словарей.
* name ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – [имя словаря](https://clickhouse.tech/docs/ru/sql-reference/dictionaries/external-dictionaries/external-dicts-dict/).
* status ([Enum8](https://clickhouse.tech/docs/ru/sql-reference/data-types/enum/)) – 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](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query_language-system-reload-dictionary), таймаут, изменение настроек словаря).
  + FAILED\_AND\_RELOADING – словарь не загружен в результате ошибки, сейчас перезагружается.
* origin ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – путь к конфигурационному файлу, описывающему словарь.
* type ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – тип размещения словаря. [Хранение словарей в памяти](https://clickhouse.tech/docs/ru/sql-reference/dictionaries/external-dictionaries/external-dicts-dict-layout/).
* key – [тип ключа](https://clickhouse.tech/docs/ru/sql-reference/dictionaries/external-dictionaries/external-dicts-dict-structure/#ext_dict_structure-key): Числовой ключ ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) или Составной ключ ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – строка вида “(тип 1, тип 2, …, тип n)”.
* attribute.names ([Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – массив [имен атрибутов](https://clickhouse.tech/docs/ru/sql-reference/dictionaries/external-dictionaries/external-dicts-dict-structure/#ext_dict_structure-attributes), предоставляемых справочником.
* attribute.types ([Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – соответствующий массив [типов атрибутов](https://clickhouse.tech/docs/ru/sql-reference/dictionaries/external-dictionaries/external-dicts-dict-structure/#ext_dict_structure-attributes), предоставляемых справочником.
* bytes\_allocated ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – объем оперативной памяти, используемый словарем.
* query\_count ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – количество запросов с момента загрузки словаря или с момента последней успешной перезагрузки.
* hit\_rate ([Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/)) – для cache-словарей – процент закэшированных значений.
* element\_count ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – количество элементов, хранящихся в словаре.
* load\_factor ([Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/)) – процент заполнения словаря (для хэшированного словаря – процент заполнения хэш-таблицы).
* source ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – текст, описывающий [источник данных](https://clickhouse.tech/docs/ru/sql-reference/dictionaries/external-dictionaries/external-dicts-dict-sources/) для словаря.
* lifetime\_min ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – минимальное [время обновления](https://clickhouse.tech/docs/ru/sql-reference/dictionaries/external-dictionaries/external-dicts-dict-lifetime/) словаря в памяти, по истечении которого RT.WideStore попытается перезагрузить словарь (если задано invalidate\_query, то только если он изменился). Задается в секундах.
* lifetime\_max ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – максимальное [время обновления](https://clickhouse.tech/docs/ru/sql-reference/dictionaries/external-dictionaries/external-dicts-dict-lifetime/) словаря в памяти, по истечении которого RT.WideStore попытается перезагрузить словарь (если задано invalidate\_query, то только если он изменился). Задается в секундах.
* loading\_start\_time ([DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/)) – время начала загрузки словаря.
* loading\_duration ([Float32](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/)) – время, затраченное на загрузку словаря.
* last\_exception ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/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─────────────────────┬─lifetime\_min─┬─lifetime\_max─┬──loading\_start\_time─┌──last\_successful\_update\_time─┬──────loading\_duration─┬─last\_exception─┐

│ dictdb │ dict │ LOADED │ dictdb.dict │ Flat │ UInt64 │ ['value\_default','value\_expression'] │ ['String','String'] │ 74032 │ 0 │ 1 │ 1 │ 0.0004887585532746823 │ ClickHouse: dictdb.dicttbl │ 0 │ 1 │ 2020-03-04 04:17:34 │ 2020-03-04 04:30:34 │ 0.002 │ │

└──────────┴──────┴────────┴─────────────┴──────┴────────┴──────────────────────────────────────┴─────────────────────┴─────────────────┴─────────────┴────────

##### system.disks

Cодержит информацию о дисках, заданных в [конфигурации сервера](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/#table_engine-mergetree-multiple-volumes_configure).

Столбцы:

* name ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя диска в конфигурации сервера.
* path ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – путь к точке монтирования в файловой системе.
* free\_space ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – свободное место на диске в байтах.
* total\_space ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – объём диска в байтах.
* keep\_free\_space ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – место, которое должно остаться 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)](https://clickhouse.tech/docs/ru/sql-reference/distributed-ddl/), которые были выполнены на кластере.

Столбцы:

* entry ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – идентификатор запроса.
* host\_name ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя хоста.
* host\_address ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – IP-адрес хоста.
* port ([UInt16](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – порт для соединения с сервером.
* status ([Enum8](https://clickhouse.tech/docs/ru/sql-reference/data-types/enum/)) – состояние запроса.
* cluster ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя кластера.
* query ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – выполненный запрос.
* initiator ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – узел, выполнивший запрос.
* query\_start\_time ([DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/)) – время начала запроса.
* query\_finish\_time ([DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/)) – время окончания запроса.
* query\_duration\_ms ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – продолжительность выполнения запроса (в миллисекундах).
* exception\_code ([Enum8](https://clickhouse.tech/docs/ru/sql-reference/data-types/enum/)) – код исключения из [ZooKeeper](https://clickhouse.tech/docs/ru/operations/tips/#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-0000000000

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-0000000000

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](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя базы данных.
* table ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя таблицы.
* data\_path ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – путь к папке с локальными файлами.
* is\_blocked ([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – флаг, указывающий на блокировку отправки локальных файлов на шарды.
* error\_count ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – количество ошибок.
* data\_files ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – количество локальных файлов в папке.
* data\_compressed\_bytes ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – размер сжатых данных в локальных файлах в байтах.
* last\_exception ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/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](https://clickhouse.tech/docs/ru/engines/table-engines/special/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](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – имя роли.
* with\_admin\_option ([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – флаг, который показывает, обладает ли enabled\_role роль привилегией ADMIN OPTION.
* is\_current ([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – флаг, который показывает, является ли enabled\_role текущей ролью текущего пользователя.
* is\_default ([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – флаг, который показывает, является ли enabled\_role ролью по умолчанию.

##### system.errors

Содержит коды ошибок с указанием количества срабатываний.

Столбцы:

* name ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – название ошибки (errorCodeToName).
* code ([Int32](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – номер кода ошибки.
* value ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – 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 RT.WideStore.

Столбцы:

* event ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя события.
* value ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – количество произошедших событий.
* description ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/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](https://clickhouse.tech/docs/ru/operations/system-tables/events/#system_tables-asynchronous_metrics) – таблица с периодически вычисляемыми метриками.
* [system.metrics](https://clickhouse.tech/docs/ru/operations/system-tables/events/#system_tables-metrics) – таблица с мгновенно вычисляемыми метриками.
* [system.metric\_log](https://clickhouse.tech/docs/ru/operations/system-tables/events/#system_tables-metric_log) – таблица фиксирующая историю значений метрик из system.metrics и system.events.
* [Мониторинг](https://clickhouse.tech/docs/ru/operations/monitoring/) – основы мониторинга в RT.WideStore.

##### 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 RT.WideStore.

* Столбцы:
* user\_name ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – название учётной записи.
* role\_name ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – роль, назначенная учетной записи пользователя.
* access\_type ([Enum8](https://clickhouse.tech/docs/ru/sql-reference/data-types/enum/)) – параметры доступа для учетной записи пользователя RT.WideStore.
* database ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – имя базы данных.
* table ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – имя таблицы.
* column ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – имя столбца, к которому предоставляется доступ.
* is\_partial\_revoke ([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – логическое значение. Показывает, были ли отменены некоторые привилегии. Возможные значения:
  + 0 – строка описывает частичный отзыв.
  + 1 – строка описывает грант.
* grant\_option ([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – разрешение предоставлено с опцией WITH GRANT OPTION, подробнее см. [GRANT](https://clickhouse.tech/docs/ru/sql-reference/statements/grant/#grant-privigele-syntax).

##### system.graphite\_retentions

Содержит информацию о том, какие параметры [graphite\_rollup](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/#server_configuration_parameters-graphite) используются в таблицах с движками [\*GraphiteMergeTree](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/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](https://github.com/ClickHouse/ClickHouse/tree/master/contrib) исходных кодов RT.WideStore.

Столбцы:

* library\_name ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – название библиотеки, к которой относится лицензия.
* license\_type ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – тип лицензии, например, Apache, MIT.
* license\_path ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – путь к файлу с текстом лицензии.
* license\_text ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/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.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](https://clickhouse.tech/docs/ru/sql-reference/data-types/date/)) – дата события.
* event\_time ([DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/)) – время события.
* event\_time\_microseconds ([DateTime64](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime64/)) – время события в микросекундах.

Пример:

**SELECT** \* **FROM** **system**.metric\_log **LIMIT** 1 FORMAT Vertical;

Row 1:

──────

event\_date: 2020-02-18

event\_time: 2020-02-18 07:15:33

milliseconds: 554

ProfileEvent\_Query: 0

ProfileEvent\_SelectQuery: 0

ProfileEvent\_InsertQuery: 0

ProfileEvent\_FileOpen: 0

ProfileEvent\_Seek: 0

ProfileEvent\_ReadBufferFromFileDescriptorRead: 1

ProfileEvent\_ReadBufferFromFileDescriptorReadFailed: 0

ProfileEvent\_ReadBufferFromFileDescriptorReadBytes: 0

ProfileEvent\_WriteBufferFromFileDescriptorWrite: 1

ProfileEvent\_WriteBufferFromFileDescriptorWriteFailed: 0

ProfileEvent\_WriteBufferFromFileDescriptorWriteBytes: 56

...

CurrentMetric\_Query: 0

CurrentMetric\_Merge: 0

CurrentMetric\_PartMutation: 0

CurrentMetric\_ReplicatedFetch: 0

CurrentMetric\_ReplicatedSend: 0

CurrentMetric\_ReplicatedChecks: 0

...

Смотрите также:

* [Настройка metric\_log](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/#metric_log) – как включить и выключить запись истории.
* [system.asynchronous\_metrics](https://clickhouse.tech/docs/ru/operations/system-tables/metric_log/#system_tables-asynchronous_metrics) – таблица с периодически вычисляемыми метриками.
* [system.events](https://clickhouse.tech/docs/ru/operations/system-tables/metric_log/#system_tables-events) – таблица с количеством произошедших событий.
* [system.metrics](https://clickhouse.tech/docs/ru/operations/system-tables/metric_log/#system_tables-metrics) – таблица с мгновенно вычисляемыми метриками.
* [Мониторинг](https://clickhouse.tech/docs/ru/operations/monitoring/) – основы мониторинга в RT.WideStore.

##### 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](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – название метрики.
* value ([Int64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – значение метрики.
* description ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – описание метрики.

Список поддерживаемых метрик смотрите в файле [src/Common/CurrentMetrics.cpp](https://github.com/ClickHouse/ClickHouse/blob/master/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](https://clickhouse.tech/docs/ru/operations/system-tables/metrics/#system_tables-asynchronous_metrics) – таблица с периодически вычисляемыми метриками.
* [system.events](https://clickhouse.tech/docs/ru/operations/system-tables/metrics/#system_tables-events) – таблица с количеством произошедших событий.
* [system.metric\_log](https://clickhouse.tech/docs/ru/operations/system-tables/metrics/#system_tables-metric_log) – таблица фиксирующая историю значений метрик из system.metrics и system.events.
* [Мониторинг](https://clickhouse.tech/docs/ru/operations/monitoring/) – основы мониторинга в RT.WideStore.

##### system.mutations

Таблица содержит информацию о ходе выполнения [мутаций](https://clickhouse.tech/docs/ru/sql-reference/statements/alter/#mutations) таблиц семейства MergeTree. Каждой команде мутации соответствует одна строка таблицы.

Столбцы:

* database ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя БД, к которой была применена мутация.
* table ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя таблицы, к которой была применена мутация.
* mutation\_id ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/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](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – команда мутации (часть запроса после ALTER TABLE [db.]table).
* create\_time ([Datetime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/)) – дата и время создания мутации.
* block\_numbers.partition\_id ([Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – для мутаций реплицированных таблиц массив содержит содержит номера партиций (по одной записи для каждой партиции). Для мутаций нереплицированных таблиц массив пустой.
* block\_numbers.number ([Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)([Int64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/))) – 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](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – массив с именами кусков данных, которые должны быть изменены для завершения мутации.
* parts\_to\_do ([Int64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – количество кусков данных, которые должны быть изменены для завершения мутации.
* is\_done ([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – 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](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя последнего куска, мутация которого не удалась.
* latest\_fail\_time ([Datetime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/)) – дата и время последней ошибки мутации.
* latest\_fail\_reason ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – причина последней ошибки мутации.

См. также:

* [Мутации](https://clickhouse.tech/docs/ru/sql-reference/statements/alter/#mutations).
* [Движок MergeTree](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/).
* [Репликация данных](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/replication/) (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](https://clickhouse.tech/docs/ru/operations/system-tables/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](https://opentracing.io/docs/overview/spans/) для выполненных запросов.

Столбцы:

* trace\_id ([UUID](https://clickhouse.tech/docs/ru/sql-reference/data-types/uuid/) – идентификатор трассировки для выполненного запроса.
* span\_id ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – идентификатор trace span.
* parent\_span\_id ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – идентификатор родительского trace span.
* operation\_name ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя операции.
* start\_time\_us ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – время начала trace span (в микросекундах).
* finish\_time\_us ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – время окончания trace span (в микросекундах).
* finish\_date ([Date](https://clickhouse.tech/docs/ru/sql-reference/data-types/date/)) – дата окончания trace span.
* attribute.names ([Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – имена [атрибутов](https://opentelemetry.io/docs/go/instrumentation/#attributes) в зависимости от trace span. Заполняются согласно рекомендациям в стандарте [OpenTelemetry](https://opentelemetry.io/).
* attribute.values ([Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – значения атрибутов в зависимости от trace span. Заполняются согласно рекомендациям в стандарте OpenTelemetry.

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](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/#server_configuration_parameters-part-log).

Содержит информацию о всех событиях, произошедших с [кусками данных](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/custom-partitioning-key/) таблиц семейства [MergeTree](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/) (например, события добавления, удаления или слияния данных).

Столбцы:

* query\_id ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – идентификатор запроса INSERT, создавшего этот кусок.
* event\_type ([Enum8](https://clickhouse.tech/docs/ru/sql-reference/data-types/enum/)) – 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](https://clickhouse.tech/docs/ru/sql-reference/statements/alter/partition/#alter_detach-partition).
  + MUTATE\_PART – изменение куска.
  + MOVE\_PART – перемещение куска между дисками.
* event\_date ([Date](https://clickhouse.tech/docs/ru/sql-reference/data-types/date/)) – дата события.
* event\_time ([DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/)) – время события.
* duration\_ms ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – длительность.
* database ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя базы данных, в которой находится кусок.
* table ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя таблицы, в которой находится кусок.
* part\_name ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя куска.
* partition\_id ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – идентификатор партиции, в которую был добавлен кусок. В столбце будет значение all, если таблица партициируется по выражению tuple().
* path\_on\_disk ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – абсолютный путь к папке с файлами кусков данных.
* rows ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – число строк в куске.
* size\_in\_bytes ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – размер куска данных в байтах.
* merged\_from ([Array(String)](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)) – массив имён кусков, из которых образован текущий кусок в результате слияния (также столбец заполняется в случае скачивания уже смерженного куска).
* bytes\_uncompressed ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – количество прочитанных не сжатых байт.
* read\_rows ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – сколько было прочитано строк при слиянии кусков.
* read\_bytes ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – сколько было прочитано байт при слиянии кусков.
* peak\_memory\_usage ([Int64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – максимальная разница между выделенной и освобождённой памятью в контексте потока.
* error ([UInt16](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – код ошибки, возникшей при текущем событии.
* exception ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/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

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](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/).

Каждая строка описывает один кусок данных.

Столбцы:

* partition ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя партиции. Что такое партиция можно узнать из описания запроса [ALTER](https://clickhouse.tech/docs/ru/sql-reference/statements/alter/#query_language_queries_alter).
* Форматы:
  + YYYYMM для автоматической схемы партиционирования по месяцам.
  + any\_string при партиционировании вручную.
* name ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя куска.
* part\_type ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/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](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/).

* active ([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – признак активности. Если кусок активен, то он используется таблицей, в противном случает он будет удален. Неактивные куски остаются после слияний.
* marks ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – количество засечек. Чтобы получить примерное количество строк в куске, умножьте marks на гранулированность индекса (обычно 8192).
* rows ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – количество строк.
* bytes\_on\_disk ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – общий размер всех файлов кусков данных в байтах.
* data\_compressed\_bytes ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – общий размер сжатой информации в куске данных. Размер всех дополнительных файлов (например, файлов с засечками) не учитывается.
* data\_uncompressed\_bytes ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – общий размер распакованной информации куска данных. Размер всех дополнительных файлов (например, файлов с засечками) не учитывается.
* marks\_bytes ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – размер файла с засечками.
* modification\_time ([DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/)) – время модификации директории с куском данных. Обычно соответствует времени создания куска.
* remove\_time ([DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/)) – время, когда кусок стал неактивным.
* refcount ([UInt32](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – количество мест, в котором кусок используется. Значение больше 2 говорит о том, что кусок участвует в запросах или в слияниях.
* min\_date ([Date](https://clickhouse.tech/docs/ru/sql-reference/data-types/date/)) – минимальное значение ключа даты в куске данных.
* max\_date ([Date](https://clickhouse.tech/docs/ru/sql-reference/data-types/date/)) – максимальное значение ключа даты в куске данных.
* min\_time ([DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/)) – минимальное значение даты и времени в куске данных.
* max\_time([DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/)) – максимальное значение даты и времени в куске данных.
* partition\_id ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – ID партиции.
* min\_block\_number ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – минимальное число кусков, из которых состоит текущий после слияния.
* max\_block\_number ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – максимальное число кусков, из которых состоит текущий после слияния.
* level ([UInt32](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – глубина дерева слияний. Если слияний не было, то level=0.
* data\_version ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – 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](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – объём памяти (в байтах), занимаемой значениями первичных ключей.
* primary\_key\_bytes\_in\_memory\_allocated ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – объём памяти (в байтах) выделенный для размещения первичных ключей.
* is\_frozen ([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – Признак, показывающий существование бэкапа партиции. 1, бэкап есть. 0, бэкапа нет. Смотрите раздел [FREEZE PARTITION](https://clickhouse.tech/docs/ru/sql-reference/statements/alter/partition/#alter_freeze-partition).
* database ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя базы данных.
* table ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя таблицы.
* engine ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя движка таблицы, без параметров.
* path ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – абсолютный путь к папке с файлами кусков данных.
* disk ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя диска, на котором находится кусок данных.
* hash\_of\_all\_files ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – значение [sipHash128](https://clickhouse.tech/docs/ru/sql-reference/functions/hash-functions/#hash_functions-siphash128) для сжатых файлов.
* hash\_of\_uncompressed\_files ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – значение [sipHash128](https://clickhouse.tech/docs/ru/sql-reference/functions/hash-functions/#hash_functions-siphash128) несжатых файлов (файлы с засечками, первичным ключом и пр.)
* uncompressed\_hash\_of\_compressed\_files ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – значение [sipHash128](https://clickhouse.tech/docs/ru/sql-reference/functions/hash-functions/#hash_functions-siphash128) данных в сжатых файлах как если бы они были разжатыми.
* delete\_ttl\_info\_min ([DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/)) – минимальное значение ключа даты и времени для правила [TTL DELETE](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/#table_engine-mergetree-ttl).
* delete\_ttl\_info\_max ([DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/)) – максимальное значение ключа даты и времени для правила [TTL DELETE](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/#table_engine-mergetree-ttl).
* move\_ttl\_info.expression ([Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – массив выражений. Каждое выражение задаёт правило [TTL MOVE](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/#table_engine-mergetree-ttl).

***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](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)([DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/))) – массив значений. Каждый элемент массива задаёт минимальное значение ключа даты и времени для правила [TTL MOVE](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/#table_engine-mergetree-ttl).
* move\_ttl\_info.max ([Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)([DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/))) – массив значений. Каждый элемент массива задаёт максимальное значение ключа даты и времени для правила [TTL MOVE](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/#table_engine-mergetree-ttl).
* bytes ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – алиас для bytes\_on\_disk.
* marks\_size ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – 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: 2d0657a16d9430824d35e327fcbd87bf

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:

* [Движок MergeTree](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/).
* [TTL для столбцов и таблиц](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/#table_engine-mergetree-ttl).

##### system.parts\_columns

Содержит информацию о кусках данных и столбцах таблиц семейства [MergeTree](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/).

Каждая строка описывает один кусок данных.

Столбцы:

* partition ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя партиции. Что такое партиция вы можете узнать из описания запроса [ALTER](https://clickhouse.tech/docs/ru/sql-reference/statements/alter/#query_language_queries_alter).

Форматы:

* YYYYMM для автоматической схемы партиционирования по месяцам.
* any\_string при партиционировании вручную.
* name ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя куска данных.
* part\_type ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/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](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/).

* active ([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – признак активности. Если кусок данных активен, то он используется таблицей, в противном случае он будет удален. Неактивные куски остаются после слияний.
* marks ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – количество засечек. Чтобы получить примерное количество строк в куске данных, умножьте marks на гранулированность индекса (обычно 8192).
* rows ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – количество строк.
* bytes\_on\_disk ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – общий размер всех файлов кусков данных в байтах.
* data\_compressed\_bytes ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – общий размер сжатой информации в куске данных. Размер всех дополнительных файлов (например, файлов с засечками) не учитывается.
* data\_uncompressed\_bytes ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – общий размер распакованной информации в куске данных. Размер всех дополнительных файлов (например, файлов с засечками) не учитывается.
* marks\_bytes ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – размер файла с засечками.
* modification\_time ([DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/)) – время модификации директории с куском данных. Обычно соответствует времени создания куска.
* remove\_time ([DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/)) – время, когда кусок данных стал неактивным.
* refcount ([UInt32](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – количество мест, в котором кусок данных используется. Значение больше 2 говорит о том, что кусок участвует в запросах или в слияниях.
* min\_date ([Date](https://clickhouse.tech/docs/ru/sql-reference/data-types/date/)) – минимальное значение ключа даты в куске данных.
* max\_date ([Date](https://clickhouse.tech/docs/ru/sql-reference/data-types/date/)) – максимальное значение ключа даты в куске данных.
* partition\_id ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – ID партиции.
* min\_block\_number ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – минимальное число кусков данных, из которых состоит текущий после слияния.
* max\_block\_number ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – максимальное число кусков данных, из которых состоит текущий после слияния.
* level ([UInt32](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – глубина дерева слияний. Если слияний не было, то level=0.
* data\_version ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – 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](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – объём памяти в байтах, занимаемой значениями первичных ключей.
* primary\_key\_bytes\_in\_memory\_allocated ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – объём памяти в байтах, выделенный для размещения первичных ключей.
* database ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя базы данных.
* table ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя таблицы.
* engine ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя движка таблицы, без параметров.
* disk\_name ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя диска, на котором находится кусок данных.
* path ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – абсолютный путь к папке с файлами кусков данных.
* column ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя столбца.
* type ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – тип столбца.
* column\_position ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – порядковый номер столбца (нумерация начинается с 1).
* default\_kind ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – тип выражения (DEFAULT, MATERIALIZED, ALIAS) для значения по умолчанию или пустая строка.
* default\_expression ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – выражение для значения по умолчанию или пустая строка.
* column\_bytes\_on\_disk ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – общий размер столбца в байтах.
* column\_data\_compressed\_bytes ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – общий размер сжатой информации в столбце в байтах.
* column\_data\_uncompressed\_bytes ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – общий размер распакованной информации в столбце в байтах.
* column\_marks\_bytes ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – размер столбца с засечками в байтах.
* bytes ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – алиас для bytes\_on\_disk.
* marks\_size ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – 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\_expression:

column\_bytes\_on\_disk: 76

column\_data\_compressed\_bytes: 28

column\_data\_uncompressed\_bytes: 2

column\_marks\_bytes: 48

Смотрите также:

* [Движок MergeTree](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/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 иницировал 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](https://clickhouse.tech/docs/ru/operations/settings/query-complexity/#settings_max_memory_usage).
* query (String) – текст запроса. Для запросов INSERT не содержит встаявляемые 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 логгирования можно изменить в секции серверной конфигурации [query\_log](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/#server_configuration_parameters-query-log).

Можно отключить логгирование настройкой [log\_queries = 0](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-log-queries). По-возможности, не отключайте логгирование, поскольку информация из таблицы важна при решении проблем.

Период сброса данных в таблицу задаётся параметром flush\_interval\_milliseconds в конфигурационной секции [query\_log](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/#server_configuration_parameters-query-log). Чтобы принудительно записать логи из буффера памяти в таблицу, используйте запрос [SYSTEM FLUSH LOGS](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query_language-system-flush_logs).

RT.WideStore не удаляет данные из таблицы автоматически.

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.

Столбцы:

* type ([Enum8](https://clickhouse.tech/docs/ru/sql-reference/data-types/enum/)) – 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](https://clickhouse.tech/docs/ru/sql-reference/data-types/date/)) – дата начала запроса.
* event\_time ([DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/)) – время начала запроса.
* event\_time\_microseconds ([DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/)) – время начала запроса с точностью до микросекунд.
* query\_start\_time ([DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/)) – время начала обработки запроса.
* query\_start\_time\_microseconds ([DateTime64](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime64/)) – время начала обработки запроса с точностью до микросекунд.
* query\_duration\_ms ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – длительность выполнения запроса в миллисекундах.
* read\_rows ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – 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 учитываюся.
* read\_bytes ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – 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 учитываюся.
* written\_rows ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – количество записанных строк для запросов INSERT. Для других запросов, значение столбца 0.
* written\_bytes ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – объём записанных данных в байтах для запросов INSERT. Для других запросов, значение столбца 0.
* result\_rows ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – количество строк в результате запроса SELECT или количество строк в запросе INSERT.
* result\_bytes ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – объём RAM в байтах, использованный для хранения результата запроса.
* memory\_usage ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – потребление RAM запросом.
* query ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – текст запроса.
* exception ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – сообщение исключения, если запрос завершился по исключению.
* exception\_code ([Int32](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – код исключения.
* stack\_trace ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – [stack trace](https://en.wikipedia.org/wiki/Stack_trace). Пустая строка, если запрос успешно завершен.
* is\_initial\_query ([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – вид запроса. Возможные значения:
  + 1 – запрос был инициирован клиентом.
  + 0 – запрос был инициирован другим запросом при выполнении распределенного запроса.
* user ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – пользователь, запустивший текущий запрос.
* query\_id ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – ID запроса.
* address ([IPv6](https://clickhouse.tech/docs/ru/sql-reference/data-types/domains/ipv6/)) – IP адрес, с которого пришел запрос.
* port ([UInt16](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – порт, с которого клиент сделал запрос
* initial\_user ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – пользователь, запустивший первоначальный запрос (для распределенных запросов).
* initial\_query\_id ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – ID родительского запроса.
* initial\_address ([IPv6](https://clickhouse.tech/docs/ru/sql-reference/data-types/domains/ipv6/)) – IP адрес, с которого пришел родительский запрос.
* initial\_port ([UInt16](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – порт, с которого клиент сделал родительский запрос.
* interface ([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – интерфейс, с которого ушёл запрос. Возможные значения:
  + 1 – TCP.
  + 2 – HTTP.
* os\_user ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя пользователя операционной системы, который запустил [clickhouse-client](https://clickhouse.tech/docs/ru/interfaces/cli/).
* client\_hostname ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя сервера, с которого присоединился [clickhouse-client](https://clickhouse.tech/docs/ru/interfaces/cli/) или другой TCP клиент.
* client\_name ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – [clickhouse-client](https://clickhouse.tech/docs/ru/interfaces/cli/) или другой TCP клиент.
* client\_revision ([UInt32](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – ревизия [clickhouse-client](https://clickhouse.tech/docs/ru/interfaces/cli/) или другого TCP клиента.
* client\_version\_major ([UInt32](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – старшая версия [clickhouse-client](https://clickhouse.tech/docs/ru/interfaces/cli/) или другого TCP клиента.
* client\_version\_minor ([UInt32](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – младшая версия [clickhouse-client](https://clickhouse.tech/docs/ru/interfaces/cli/) или другого TCP клиента.
* client\_version\_patch ([UInt32](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – патч [clickhouse-client](https://clickhouse.tech/docs/ru/interfaces/cli/) или другого TCP клиента.
* http\_method ([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – HTTP метод, инициировавший запрос. Возможные значения:
  + 0 – запрос запущен с интерфейса TCP.
  + 1 – GET.
  + 2 – POST.
* http\_user\_agent ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – HTTP заголовок UserAgent.
* quota\_key ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – «ключ квоты» из настроек [квот](https://clickhouse.tech/docs/ru/operations/system-tables/quotas/) (см. keyed).
* revision ([UInt32](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – ревизия ClickHouse.
* thread\_numbers ([Array(UInt32)](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)) – количество потоков, участвующих в обработке запросов.
* ProfileEvents.Names ([Array(String)](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)) – счетчики для изменения различных метрик. Описание метрик можно получить из таблицы [system.events](https://clickhouse.tech/docs/ru/operations/system-tables/query_log/#system_tables-events)(#system\_tables-events
* ProfileEvents.Values ([Array(UInt64)](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)) – метрики, перечисленные в столбце ProfileEvents.Names.
* Settings.Names ([Array(String)](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)) – имена настроек, которые меняются, когда клиент выполняет запрос. Чтобы разрешить логирование изменений настроек, установите параметр log\_query\_settings равным 1.
* Settings.Values ([Array(String)](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)) – значения настроек, которые перечислены в столбце Settings.Names.

Пример:

**SELECT** \* **FROM** **system**.query\_log **LIMIT** 1 \**G**

Row 1:

──────

type: QueryStart

event\_date: 2020-09-11

event\_time: 2020-09-11 10:08:17

event\_time\_microseconds: 2020-09-11 10:08:17.063321

query\_start\_time: 2020-09-11 10:08:17

query\_start\_time\_microseconds: 2020-09-11 10:08:17.063321

query\_duration\_ms: 0

read\_rows: 0

read\_bytes: 0

written\_rows: 0

written\_bytes: 0

result\_rows: 0

result\_bytes: 0

memory\_usage: 0

current\_database: default

query: INSERT INTO test1 VALUES

exception\_code: 0

exception:

stack\_trace:

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

thread\_ids: []

ProfileEvents.Names: []

ProfileEvents.Values: []

Settings.Names: ['use\_uncompressed\_cache','load\_balancing','log\_queries','max\_memory\_usage','allow\_introspection\_functions']

Settings.Values: ['0','random','1','10000000000','1']

Смотрите также:

* [system.query\_thread\_log](https://clickhouse.tech/docs/ru/operations/system-tables/query_thread_log/#system_tables-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](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/#server_configuration_parameters-query_thread_log) в конфигурации сервера.
2. Установите значение [log\_query\_threads](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-log-query-threads) равным 1.

Интервал сброса данных в таблицу задаётся параметром flush\_interval\_milliseconds в разделе настроек сервера [query\_thread\_log](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/#server_configuration_parameters-query_thread_log). Чтобы принудительно записать логи из буфера памяти в таблицу, используйте запрос [SYSTEM FLUSH LOGS](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query_language-system-flush_logs).

ClickHouse не удаляет данные из таблицы автоматически. Подробности в разделе [Введение](https://clickhouse.tech/docs/ru/operations/system-tables/query_thread_log/#system-tables-introduction).

Столбцы:

* event\_date ([Date](https://clickhouse.tech/docs/ru/sql-reference/data-types/date/)) – дата завершения выполнения запроса потоком.
* event\_time ([DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/)) – дата и время завершения выполнения запроса потоком.
* event\_time\_microseconds ([DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/)) – дата и время завершения выполнения запроса потоком с точностью до микросекунд.
* query\_start\_time ([DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/)) – время начала обработки запроса.
* query\_start\_time\_microseconds ([DateTime64](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime64/)) – время начала обработки запроса с точностью до микросекунд.
* query\_duration\_ms ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – длительность обработки запроса в миллисекундах.
* read\_rows ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – количество прочитанных строк.
* read\_bytes ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – количество прочитанных байтов.
* written\_rows ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – количество записанных строк для запросов INSERT. Для других запросов, значение столбца 0.
* written\_bytes ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – объём записанных данных в байтах для запросов INSERT. Для других запросов, значение столбца 0.
* memory\_usage ([Int64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – разница между выделенной и освобождённой памятью в контексте потока.
* peak\_memory\_usage ([Int64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – максимальная разница между выделенной и освобождённой памятью в контексте потока.
* thread\_name ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя потока.
* thread\_id ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – tid (ID потока операционной системы).
* master\_thread\_id ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – tid (ID потока операционной системы) главного потока.
* query ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – текст запроса.
* is\_initial\_query ([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – вид запроса. Возможные значения:
  + 1 – запрос был инициирован клиентом.
  + 0 – запрос был инициирован другим запросом при распределенном запросе.
* user ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – пользователь, запустивший текущий запрос.
* query\_id ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – ID запроса.
* address ([IPv6](https://clickhouse.tech/docs/ru/sql-reference/data-types/domains/ipv6/)) – IP адрес, с которого пришел запрос.
* port ([UInt16](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – порт, с которого пришел запрос.
* initial\_user ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – пользователь, запустивший первоначальный запрос (для распределенных запросов).
* initial\_query\_id ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – ID родительского запроса.
* initial\_address ([IPv6](https://clickhouse.tech/docs/ru/sql-reference/data-types/domains/ipv6/)) – IP адрес, с которого пришел родительский запрос.
* initial\_port ([UInt16](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – порт, пришел родительский запрос.
* interface ([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – интерфейс, с которого ушёл запрос. Возможные значения:
  + 1 – TCP.
  + 2 – HTTP.
* os\_user ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя пользователя в OS, который запустил [clickhouse-client](https://clickhouse.tech/docs/ru/interfaces/cli/).
* client\_hostname ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – hostname клиентской машины, с которой присоединился [clickhouse-client](https://clickhouse.tech/docs/ru/interfaces/cli/) или другой TCP клиент.
* client\_name ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – [clickhouse-client](https://clickhouse.tech/docs/ru/interfaces/cli/) или другой TCP клиент.
* client\_revision ([UInt32](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – ревизия [clickhouse-client](https://clickhouse.tech/docs/ru/interfaces/cli/) или другого TCP клиента.
* client\_version\_major ([UInt32](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – старшая версия [clickhouse-client](https://clickhouse.tech/docs/ru/interfaces/cli/) или другого TCP клиента.
* client\_version\_minor ([UInt32](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – младшая версия [clickhouse-client](https://clickhouse.tech/docs/ru/interfaces/cli/) или другого TCP клиента.
* client\_version\_patch ([UInt32](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – патч [clickhouse-client](https://clickhouse.tech/docs/ru/interfaces/cli/) или другого TCP клиента.
* http\_method ([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – HTTP метод, инициировавший запрос. Возможные значения:
  + 0 – запрос запущен с интерфейса TCP.
  + 1 – GET.
  + 2 – POST.
* http\_user\_agent ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – HTTP заголовок UserAgent.
* quota\_key ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – «ключ квоты» из настроек [квот](https://clickhouse.tech/docs/ru/operations/system-tables/quotas/) (см. keyed).
* revision ([UInt32](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – ревизия RT.WideStore.
* ProfileEvents.Names ([Array(String)](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)) – счетчики для изменения различных метрик для данного потока. Описание метрик можно получить из таблицы [system.events](https://clickhouse.tech/docs/ru/operations/system-tables/query_thread_log/#system_tables-events).
* ProfileEvents.Values ([Array(UInt64)](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)) – метрики для данного потока, перечисленные в столбце ProfileEvents.Names.

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.Names: ['Query','InsertQuery','FileOpen','WriteBufferFromFileDescriptorWrite','WriteBufferFromFileDescriptorWriteBytes','ReadCompressedBytes','CompressedReadBufferBlocks','CompressedReadBufferBytes','IOBufferAllocs','IOBufferAllocBytes','FunctionExecute','CreatedWriteBufferOrdinary','DiskWriteElapsedMicroseconds','NetworkReceiveElapsedMicroseconds','NetworkSendElapsedMicroseconds','InsertedRows','InsertedBytes','SelectedRows','SelectedBytes','MergeTreeDataWriterRows','MergeTreeDataWriterUncompressedBytes','MergeTreeDataWriterCompressedBytes','MergeTreeDataWriterBlocks','MergeTreeDataWriterBlocksAlreadySorted','ContextLock','RWLockAcquiredReadLocks','RealTimeMicroseconds','UserTimeMicroseconds','SoftPageFaults','OSCPUVirtualTimeMicroseconds','OSWriteBytes','OSReadChars','OSWriteChars']

ProfileEvents.Values: [1,1,11,11,591,148,3,71,29,6533808,1,11,72,18,47,1,12,1,12,1,12,189,1,1,10,2,70853,2748,49,2747,45056,422,1520]

Смотрите также:

* [system.query\_log](https://clickhouse.tech/docs/ru/operations/system-tables/query_log/#system_tables-query_log) – описание системной таблицы query\_log, которая содержит общую информацию о выполненных запросах.

##### 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](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя квоты.
* duration ([UInt32](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – длина временного интервала для расчета потребления ресурсов, в секундах.
* is\_randomized\_interval ([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – 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](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/))) – максимальное число запросов.
* max\_errors ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/))) – максимальное количество ошибок.
* max\_result\_rows ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/))) – максимальное количество строк результата.
* max\_result\_bytes ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/))) – максимальный объем оперативной памяти в байтах, используемый для хранения результата запроса.
* max\_read\_rows ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/))) – максимальное количество строк, считываемых из всех таблиц и табличных функций, участвующих в запросе.
* max\_read\_bytes ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/))) – максимальное количество байтов, считываемых из всех таблиц и табличных функций, участвующих в запросе.
* max\_execution\_time ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/))) – максимальное время выполнения запроса, в секундах.

##### system.quota\_usage

Использование квоты текущим пользователем: сколько используется и сколько осталось.

Столбцы:

* quota\_name ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя квоты.
* quota\_key([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – значение ключа. Например, если keys = ip\_address, quota\_key может иметь значение '192.168.1.1'.
* start\_time([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/))) – время начала расчета потребления ресурсов.
* end\_time([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/))) – время окончания расчета потребления ресурс
* duration ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/))) – длина временного интервала для расчета потребления ресурсов, в секундах.
* queries ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/))) – общее количество запросов на этом интервале.
* max\_queries ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/))) – максимальное количество запросов.
* errors ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/))) – число запросов, вызвавших ошибки.
* max\_errors ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/))) – максимальное число ошибок.
* result\_rows ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/))) – общее количество строк результата.
* max\_result\_rows ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/))) – максимальное количество строк результата.
* result\_bytes ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/))) – объем оперативной памяти в байтах, используемый для хранения результата запроса.
* max\_result\_bytes ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/))) – максимальный объем оперативной памяти, используемый для хранения результата запроса, в байтах.
* read\_rows ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/))) – общее число исходных строк, считываемых из таблиц для выполнения запроса на всех удаленных серверах.
* max\_read\_rows ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/))) – максимальное количество строк, считываемых из всех таблиц и табличных функций, участвующих в запросах.
* read\_bytes ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/))) – общее количество байт, считанных из всех таблиц и табличных функций, участвующих в запросах.
* max\_read\_bytes ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/))) – максимальное количество байт, считываемых из всех таблиц и табличных функций.
* execution\_time ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/))) – общее время выполнения запроса, в секундах.
* max\_execution\_time ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/))) – максимальное время выполнения запроса.

Смотрите также:

* [SHOW QUOTA](https://clickhouse.tech/docs/ru/sql-reference/statements/show/#show-quota-statement).

##### system.quotas

Содержит информацию о [квотах](https://clickhouse.tech/docs/ru/operations/system-tables/quotas/).

Столбцы:

* name ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя квоты.
* id ([UUID](https://clickhouse.tech/docs/ru/sql-reference/data-types/uuid/)) – ID квоты.
* storage([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – хранилище квот. Возможные значения: "users.xml", если квота задана в файле users.xml, "disk" – если квота задана в SQL-запросе.
* keys ([Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)([Enum8](https://clickhouse.tech/docs/ru/sql-reference/data-types/enum/))) – 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](https://clickhouse.tech/docs/ru/interfaces/cli/), 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](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/))) – длины временных интервалов для расчета потребления ресурсов, в секундах.
* apply\_to\_all ([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – 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](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – список имен пользователей/[ролей](https://clickhouse.tech/docs/ru/operations/access-rights/#role-management) к которым применяется квота.
* apply\_to\_except ([Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – список имен пользователей/ролей к которым квота применяться не должна.

Смотрите также:

* [SHOW QUOTAS](https://clickhouse.tech/docs/ru/sql-reference/statements/show/#show-quotas-statement).

##### system.quotas\_usage

Использование квот всеми пользователями.

Столбцы:

* quota\_name ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя квоты.
* quota\_key ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – ключ квоты.
* is\_current ([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – квота используется для текущего пользователя.
* start\_time ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/)))) – время начала расчета потребления ресурсов.
* end\_time ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/)))) – время окончания расчета потребления ресурсов.
* duration ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([UInt32](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/))) – длина временного интервала для расчета потребления ресурсов, в секундах.
* queries ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/))) – общее количество запросов на этом интервале.
* max\_queries ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/))) – максимальное число запросов.
* errors ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/))) – число запросов, вызвавших ошибки.
* max\_errors ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/))) – максимальное число ошибок.
* result\_rows ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/))) – Общее количество строк, приведенных в результате.
* max\_result\_rows ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/))) – максимальное количество исходных строк, считываемых из таблиц.
* result\_bytes ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/))) – объем оперативной памяти в байтах, используемый для хранения результата запроса.
* max\_result\_bytes ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/))) – максимальный объем оперативной памяти, используемый для хранения результата запроса, в байтах.
* read\_rows ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/))) – общее число исходных строк, считываемых из таблиц для выполнения запроса на всех удаленных серверах.
* max\_read\_rows ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/))) – максимальное количество строк, считываемых из всех таблиц и табличных функций, участвующих в запросах.
* read\_bytes ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/))) – общее количество байт, считанных из всех таблиц и табличных функций, участвующих в запросах.
* max\_read\_bytes ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/))) – максимальное количество байт, считываемых из всех таблиц и табличных функций.
* execution\_time ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/))) – общее время выполнения запроса, в секундах.
* max\_execution\_time ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/))) – максимальное время выполнения запроса.

Смотрите также:

* [SHOW QUOTA](https://clickhouse.tech/docs/ru/sql-reference/statements/show/#show-quota-statement).

##### 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** = 'visits' FORMAT Vertical

Row 1:

──────

database: merge

table: visits

engine: ReplicatedCollapsingMergeTree

is\_leader: 1

can\_become\_leader: 1

is\_readonly: 0

is\_session\_expired: 0

future\_parts: 1

parts\_to\_check: 0

zookeeper\_path: /clickhouse/tables/01-06/visits

replica\_name: example01-06-1.widestore.ru

replica\_path: /clickhouse/tables/01-06/visits/replicas/example01-06-1.widestore.ru

columns\_version: 9

queue\_size: 1

inserts\_in\_queue: 0

merges\_in\_queue: 1

part\_mutations\_in\_queue: 0

queue\_oldest\_time: 2020-02-20 08:34:30

inserts\_oldest\_time: 0000-00-00 00:00:00

merges\_oldest\_time: 2020-02-20 08:34:30

part\_mutations\_oldest\_time: 0000-00-00 00:00:00

oldest\_part\_to\_get:

oldest\_part\_to\_merge\_to: 20200220\_20284\_20840\_7

oldest\_part\_to\_mutate\_to:

log\_max\_index: 596273

log\_pointer: 596274

last\_queue\_update: 2020-02-20 08:34:32

absolute\_delay: 0

total\_replicas: 2

active\_replicas: 2

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) – когда очередь обновлялась в последний раз.
* absolute\_delay (UInt64) – насколько велика задержка в секундах у 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.

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](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя базы данных.
* table ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя таблицы.
* elapsed ([Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/)) – время, прошедшее от момента начала скачивания куска, в секундах.
* progress ([Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/)) – доля выполненной работы от 0 до 1.
* result\_part\_name ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя скачиваемого куска.
* result\_part\_path ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – абсолютный путь к скачиваемому куску.
* partition\_id ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – идентификатор партиции.
* total\_size\_bytes\_compressed ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – общий размер сжатой информации в скачиваемом куске в байтах.
* bytes\_read\_compressed ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – размер сжатой информации, считанной из скачиваемого куска, в байтах.
* source\_replica\_path ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – абсолютный путь к исходной реплике.
* source\_replica\_hostname ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя хоста исходной реплики.
* source\_replica\_port ([UInt16](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – номер порта исходной реплики.
* interserver\_scheme ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя межсерверной схемы.
* URI ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – универсальный идентификатор ресурса.
* to\_detached ([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – флаг, указывающий на использование выражения TO DETACHED в текущих фоновых операциях.
* thread\_id ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – 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](https://clickhouse.tech/docs/ru/operations/sql-reference/statements/system/#query-language-system-replicated).

##### system.replication\_queue

Содержит информацию о задачах из очередей репликации, хранящихся в ZooKeeper, для таблиц семейства ReplicatedMergeTree.

Столбцы:

* database ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя базы данных.
* table ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя таблицы.
* replica\_name ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя реплики в ZooKeeper. Разные реплики одной и той же таблицы имеют различные имена.
* position ([UInt32](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – позиция задачи в очереди.
* node\_name ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя узла в ZooKeeper.
* type ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – тип задачи в очереди: GET\_PARTS, MERGE\_PARTS, DETACH\_PARTS, DROP\_PARTS или MUTATE\_PARTS.
* create\_time ([Datetime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/)) – дата и время отправки задачи на выполнение.
* required\_quorum ([UInt32](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – количество реплик, ожидающих завершения задачи, с подтверждением о завершении. Этот столбец актуален только для задачи GET\_PARTS.
* source\_replica ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя исходной реплики.
* new\_part\_name ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя нового куска.
* parts\_to\_merge ([Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/) ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – имена кусков, которые требуется смержить или обновить.
* is\_detach ([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – флаг, указывающий на присутствие в очереди задачи DETACH\_PARTS.
* is\_currently\_executing ([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – флаг, указывающий на выполнение конкретной задачи на данный момент.
* num\_tries ([UInt32](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – количество неудачных попыток выполнить задачу.
* last\_exception ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – текст сообщения о последней возникшей ошибке, если таковые имеются.
* last\_attempt\_time ([Datetime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/)) – дата и время последней попытки выполнить задачу.
* num\_postponed ([UInt32](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – количество отложенных задач.
* postpone\_reason ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – причина, по которой была отложена задача.
* last\_postpone\_time ([Datetime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/)) – дата и время, когда была отложена задача в последний раз.
* merge\_type ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/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.widestore.rt.ru

position: 15

node\_name: queue-0009325559

type: MERGE\_PARTS

create\_time: 2020-12-07 14:04:21

required\_quorum: 0

source\_replica: mtgiga001-1t.widestore.rt.ru

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](https://clickhouse.tech/docs/ru/operations/sql-reference/statements/system.md/#query-language-system-replicated).

##### system.role\_grants

Содержит [гранты](https://clickhouse.tech/docs/ru/sql-reference/statements/grant/) ролей для пользователей и ролей. Чтобы добавить записи в эту таблицу, используйте команду GRANT role TO user.

Столбцы:

* user\_name ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – имя пользователя.
* role\_name ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – имя роли.
* granted\_role\_name ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя роли, назначенной для роли role\_name. Чтобы назначить одну роль другой используйте GRANT role1 TO role2.
* granted\_role\_is\_default ([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – 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](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – 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](https://clickhouse.tech/docs/ru/operations/access-rights/#role-management).

Столбцы:

* name ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя роли.
* id ([UUID](https://clickhouse.tech/docs/ru/sql-reference/data-types/uuid/)) – ID роли.
* storage ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – путь к хранилищу ролей. Настраивается в параметре access\_control\_path.

Смотрите также:

* [SHOW ROLES](https://clickhouse.tech/docs/ru/sql-reference/statements/show/#show-roles-statement).

##### 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](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя политики строк.
* short\_name ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/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](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя базы данных.
* table ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя таблицы.
* id ([UUID](https://clickhouse.tech/docs/ru/sql-reference/data-types/uuid/)) – ID политики строк.
* storage ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя каталога, в котором хранится политика строк.
* select\_filter ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – условие, которое используется для фильтрации строк.
* is\_restrictive ([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – показывает, ограничивает ли политика строк доступ к строкам, подробнее см. [CREATE ROW POLICY](https://clickhouse.tech/docs/ru/sql-reference/statements/create/row-policy/#create-row-policy-as). Значения:
* 0 – политика строк определяется с помощью условия 'AS PERMISSIVE'.
* 1 – политика строк определяется с помощью условия 'AS RESTRICTIVE'.
* apply\_to\_all ([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – показывает, что политики строк заданы для всех ролей и/или пользователей.
* apply\_to\_list ([Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – список ролей и/или пользователей, к которым применяется политика строк.
* apply\_to\_except ([Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – политики строк применяются ко всем ролям и/или пользователям, за исключением перечисленных.

Смотрите также:

* [SHOW POLICIES](https://clickhouse.tech/docs/ru/sql-reference/statements/show/#show-policies-statement).

##### system.settings

Содержит информацию о сессионных настройках для текущего пользователя.

Столбцы:

* name ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя настройки.
* value ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – значение настройки.
* changed ([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – показывает, изменена ли настройка по отношению к значению по умолчанию.
* description ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – краткое описание настройки.
* min ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – минимальное значение настройки, если задано [ограничение](https://clickhouse.tech/docs/ru/operations/settings/constraints-on-settings/#constraints-on-settings). Если нет, то поле содержит [NULL](https://clickhouse.tech/docs/ru/sql-reference/syntax/#null-literal).
* max ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – максимальное значение настройки, если задано [ограничение](https://clickhouse.tech/docs/ru/operations/settings/constraints-on-settings/#constraints-on-settings). Если нет, то поле содержит [NULL](https://clickhouse.tech/docs/ru/sql-reference/syntax/#null-literal).
* readonly ([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – 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'

Cм. также:

* [Настройки](https://clickhouse.tech/docs/ru/operations/settings/#settings).
* [Разрешения для запросов](https://clickhouse.tech/docs/ru/operations/settings/permissions-for-queries/#settings_readonly).
* [Ограничения для значений настроек](https://clickhouse.tech/docs/ru/operations/settings/constraints-on-settings/).

##### system.settings\_profile\_elements

Описывает содержимое профиля настроек:

* Ограничения.
* Роли и пользователи, к которым применяется настройка.
* Родительские профили настроек.

Столбцы:

* profile\_name ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – имя профиля настроек.
* user\_name ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – имя пользователя.
* role\_name ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – имя роли.
* index ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – порядковый номер элемента профиля настроек.
* setting\_name ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – имя настройки.
* value ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – значение настройки.
* min ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – минимальное значение настройки. NULL если не задано.
* max ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – максимальное значение настройки. NULL если не задано.
* readonly ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges))) – профиль разрешает только запросы на чтение.
* inherit\_profile ([Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – родительский профиль для данного профиля настроек. NULL 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](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя профиля настроек.
* id ([UUID](https://clickhouse.tech/docs/ru/sql-reference/data-types/uuid/)) – ID профиля настроек.
* storage ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – путь к хранилищу профилей настроек. Настраивается в параметре access\_control\_path.
* num\_elements ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – число элементов для этого профиля в таблице system.settings\_profile\_elements.
* apply\_to\_all ([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – признак, который показывает, что параметры профиля заданы для всех ролей и/или пользователей.
* apply\_to\_list ([Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – список ролей и/или пользователей, к которым применяется профиль настроек.
* apply\_to\_except ([Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – профиль настроек применяется ко всем ролям и/или пользователям, за исключением перечисленных.

Смотрите также:

* [SHOW PROFILES](https://clickhouse.tech/docs/ru/sql-reference/statements/show/#show-profiles-statement).

##### system.stack\_trace

Содержит трассировки стека всех серверных потоков. Позволяет разработчикам анализировать состояние сервера.

Для анализа логов используйте [функции интроспекции](https://clickhouse.tech/docs/ru/sql-reference/functions/introspection/): addressToLine, addressToSymbol и demangle.

Столбцы:

* thread\_id ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – идентификатор потока.
* query\_id ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – идентификатор запроса. Может быть использован для получения подробной информации о выполненном запросе из системной таблицы [query\_log](https://clickhouse.tech/docs/ru/operations/system-tables/stack_trace/#system_tables-query_log).
* trace ([Array(UInt64)](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)) – [трассировка стека](https://en.wikipedia.org/wiki/Stack_trace). Представляет собой список физических адресов, по которым расположены вызываемые методы.

Пример:

Включение функций интроспекции:

**SET** allow\_introspection\_functions = 1;

Получение символов из объектных файлов RT.WideStore:

**WITH** arrayMap(x -> demangle(addressToSymbol(x)), trace) **AS** **all** **SELECT** thread\_id, query\_id, arrayStringConcat(**all**, '\n') **AS** res **FROM** **system**.stack\_trace **LIMIT** 1 \**G**

Row 1:

──────

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 RT.WideStore:

**WITH** arrayMap(x -> addressToLine(x), trace) **AS** **all**, arrayFilter(x -> x **LIKE** '%/dbms/%', **all**) **AS** dbms **SELECT** thread\_id, query\_id, arrayStringConcat(notEmpty(dbms) ? dbms : **all**, '\n') **AS** res **FROM** **system**.stack\_trace **LIMIT** 1 \**G**

Row 1:

──────

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

См. также:

* [Функции интроспекции](https://clickhouse.tech/docs/ru/sql-reference/functions/introspection/) – что такое функции интроспекции и как их использовать.
* [system.trace\_log](https://clickhouse.tech/docs/ru/operations/system-tables/trace_log/#system_tables-trace_log) – содержит трассировки стека, собранные профилировщиком выборочных запросов.
* [arrayMap](https://clickhouse.tech/docs/ru/sql-reference/functions/array-functions/#array-map) – описание и пример использования функции arrayMap.
* [arrayFilter](https://clickhouse.tech/docs/ru/sql-reference/functions/array-functions/#array-filter) – описание и пример использования функции arrayFilter.

##### system.storage\_policies

Содержит информацию о политиках хранения и томах, заданных в [конфигурации сервера](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/#table_engine-mergetree-multiple-volumes_configure).

Столбцы:

* policy\_name ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя политики хранения.
* volume\_name ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя тома, который содержится в политике хранения.
* volume\_priority ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – порядковый номер тома согласно конфигурации.
* disks ([Array(String)](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)) – имена дисков, содержащихся в политике хранения.
* max\_data\_part\_size ([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – 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](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – значение настройки prefer\_not\_to\_merge. Если данная настройка включена, то слияние данных, хранящихся на данном томе, не допускается. Это позволяет контролировать работу RT.WideStore 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](https://clickhouse.tech/docs/ru/operations/system-tables/table_engines/table_engines/mergetree/#table_engine-mergetree-data_skipping-indexes).
* supports\_ttl (UInt8) – флаг, показывающий поддержку [TTL](https://clickhouse.tech/docs/ru/operations/system-tables/table_engines/table_engines/mergetree/#table_engine-mergetree-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](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/replication/).
* 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 │ 0 │ 0 │

│ MergeTree │ 1 │ 1 │ 1 │ 1 │ 0 │ 0 │

│ ReplicatedCollapsingMergeTree │ 1 │ 1 │ 1 │ 1 │ 1 │ 1 │

└───────────────────────────────┴───────────────────┴───────────────────────────┴─────────────────────┴──────────────┴──────────────────────┴────────────────────────┘

Смотрите также**:**

* [Секции движка](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/#mergetree-query-clauses) семейства MergeTree.
* [Настройки](https://clickhouse.tech/docs/ru/engines/table-engines/integrations/kafka/#table_engine-kafka-creating-a-table) Kafka.
* [Настройки](https://clickhouse.tech/docs/ru/engines/table-engines/special/join/#join-limitations-and-settings) Join.

##### system.tables

Содержит метаданные каждой таблицы, о которой знает сервер. Отсоединённые таблицы не отображаются в system.tables.

Эта таблица содержит следующие столбцы (тип столбца показан в скобках):

* 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](https://clickhouse.tech/docs/ru/engines/table-engines/special/materializedview/), созданные на базе текущей таблицы).
* create\_table\_query (String) – запрос, которым создавалась таблица.
* engine\_full (String) – параметры табличного движка.
* partition\_key (String) – ключ партиционирования 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](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/#table_engine-mergetree-multiple-volumes),
  + [Distributed](https://clickhouse.tech/docs/ru/engines/table-engines/special/distributed/#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).

Таблица system.tables используется при выполнении запроса SHOW TABLES.

##### 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 RT.WideStore.
* 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.trace\_log

Содержит экземпляры трассировки стека адресов вызова, собранные с помощью семплирующего профайлера запросов.

ClickHouse создает эту таблицу когда утсановлена настройка [trace\_log](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/#server_configuration_parameters-trace_log) в конфигурационном файле сервереа. А также настройки [query\_profiler\_real\_time\_period\_ns](https://clickhouse.tech/docs/ru/operations/settings/settings/#query_profiler_real_time_period_ns) и [query\_profiler\_cpu\_time\_period\_ns](https://clickhouse.tech/docs/ru/operations/settings/settings/#query_profiler_cpu_time_period_ns).

Для анализа stack traces, используйте функции интроспекции addressToLine, addressToSymbol и demangle.

Столбцы:

* event\_date([Date](https://clickhouse.tech/docs/ru/sql-reference/data-types/date/)) – дата в момент снятия экземпляра стэка адресов вызова.
* event\_time([DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/)) – дата и время в момент снятия экземпляра стэка адресов вызова.
* event\_time\_microseconds ([DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/)) – дата и время в момент снятия экземпляра стэка адресов вызова с точностью до микросекунд.
* revision([UInt32](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – ревизия сборки сервера RT.WideStore.
* 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`.
* timer\_type([Enum8](https://clickhouse.tech/docs/ru/sql-reference/data-types/enum/)) – тип таймера:
  + `Real` означает wall-clock время.
  + `CPU` означает относительное CPU время.
* thread\_number([UInt32](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)) – идентификатор треда.
* query\_id([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – идентификатор запроса который может быть использован для получения деталей о запросе из таблицы [query\_log](https://clickhouse.tech/docs/ru/operations/system-tables/query_log/#system_tables-query_log) system table.
* trace([Array(UInt64)](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)) – трассировка стека адресов вызова в момент семплирования. Каждый элемент массива – это адрес виртуальной памяти внутри процесса сервера RT.WideStore.

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: [371912858,371912789,371798468,371799717,371801313,371790250,624462773,566365041,566440261,566445834,566460071,566459914,566459842,566459580,566459469,566459389,566459341,566455774,371993941,371988245,372158848,372187428,372187309,372187093,372185478,140222123165193,140222122205443]

size: 5244400

##### system.users

Содержит список [аккаунтов пользователей](https://clickhouse.tech/docs/ru/operations/access-rights/#user-account-management), настроенных на сервере.

Столбцы:

* name ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – имя пользователя.
* id ([UUID](https://clickhouse.tech/docs/ru/sql-reference/data-types/uuid/)) – ID пользователя.
* storage ([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – путь к хранилищу пользователей. Настраивается в параметре access\_control\_path.
* auth\_type ([Enum8](https://clickhouse.tech/docs/ru/sql-reference/data-types/enum/)('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](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)) – параметры аутентификации в формате JSON, зависят от auth\_type.
* host\_ip ([Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – IP-адреса хостов, которым разрешено подключаться к серверу RT.WideStore.
* host\_names ([Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – имена хостов, которым разрешено подключаться к серверу RT.WideStore.
* host\_names\_regexp ([Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – регулярное выражение для имен хостов, которым разрешено подключаться к серверу RT.WideStore.
* host\_names\_like ([Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – имена хостов, которым разрешено подключаться к серверу RT.WideStore, заданные с помощью предиката LIKE.
* default\_roles\_all ([UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges)) – показывает, что все предоставленные роли установлены для пользователя по умолчанию.
* default\_roles\_list ([Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – список предоставленных ролей по умолчанию.
* default\_roles\_except ([Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/))) – все предоставленные роли задаются по умолчанию, за исключением перечисленных.

Смотрите также:

* [SHOW USERS](https://clickhouse.tech/docs/ru/sql-reference/statements/show/#show-users-statement).

##### 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.widestore.rt.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.widestore.rt.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

ПРИЛОЖЕНИЕ 3. ОПИСАНИЕ ПАРАМЕТРОВ ОГРАНИЧЕНИЯ НА СЛОЖНОСТЬ ЗАПРОСОВ

##### 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](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/#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](https://github.com/ClickHouse/ClickHouse/blob/master/src/Core/Settings.h#L288). По умолчанию размер не ограничен (max\_memory\_usage\_for\_user = 0).

Смотрите также описание настройки [max\_memory\_usage](https://clickhouse.tech/docs/ru/operations/settings/query-complexity/#settings_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 обьединения полученных результатов. Например, кластер состоит из 2 шард и каждый из них хранит таблицу с 100 строк. Тогда распределнный 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 обьединения полученных результатов. Например, кластер состоит из 2 шард и каждый из них хранит таблицу со 100 байтами. Тогда распределнный 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 приближённо. Качество такого приближённого 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](https://clickhouse.tech/docs/ru/sql-reference/statements/select/group-by/#select-group-by-in-external-memory).

Возможные значения:

* Максимальный объём RAM (в байтах), который может использовать отдельная операция [GROUP BY](https://clickhouse.tech/docs/ru/sql-reference/statements/select/group-by/#select-group-by-clause).
* 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](https://clickhouse.tech/docs/ru/operations/settings/query-complexity/#setting-max_result_rows), кратно [max\_block\_size](https://clickhouse.tech/docs/ru/operations/settings/settings/#setting-max_block_size) и зависит от [max\_threads](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-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 (несжатых 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](https://clickhouse.tech/docs/ru/sql-reference/statements/select/join/#select-join) и к движку таблиц [Join](https://clickhouse.tech/docs/ru/engines/table-engines/special/join/).

Если запрос содержит несколько JOIN, то RT.WideStore проверяет значение настройки для каждого промежуточного результата.

При достижении предела RT.WideStore может выполнять различные действия. Используйте настройку [join\_overflow\_mode](https://clickhouse.tech/docs/ru/operations/settings/query-complexity/#settings-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](https://clickhouse.tech/docs/ru/sql-reference/statements/select/join/#select-join) и к движку таблиц [Join](https://clickhouse.tech/docs/ru/engines/table-engines/special/join/).

Если запрос содержит несколько JOIN, то RT.WideStore проверяет значение настройки для каждого промежуточного результата.

При достижении предела RT.WideStore может выполнять различные действия. Используйте настройку [join\_overflow\_mode](https://clickhouse.tech/docs/ru/operations/settings/query-complexity/#settings-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 RT.WideStore выполняет при достижении любого из следующих ограничений для JOIN:

* [max\_bytes\_in\_join](https://clickhouse.tech/docs/ru/operations/settings/query-complexity/#settings-max_bytes_in_join)
* [max\_rows\_in\_join](https://clickhouse.tech/docs/ru/operations/settings/query-complexity/#settings-max_rows_in_join)

Возможные значения:

* THROW – RT.WideStore генерирует исключение и прерывает операцию.
* BREAK – RT.WideStore прерывает операцию, но не генерирует исключение.

Значение по умолчанию – THROW.

Смотрите также:

* [Секция JOIN](https://clickhouse.tech/docs/ru/sql-reference/statements/select/join/#select-join).
* [Движок таблиц Join](https://clickhouse.tech/docs/ru/engines/table-engines/special/join/).

##### max\_partitions\_per\_insert\_block

Ограничивает максимальное количество партиций в одном вставленном блоке.

* Положительное целое число.
* 0 – неограниченное количество разделов.

Значение по умолчанию: 100.

При вставке данных, RT.WideStore вычисляет количество партиций во вставленном блоке. Если число партиций больше, чем max\_partitions\_per\_insert\_block, RT.WideStore 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)».

ПРИЛОЖЕНИЕ 4. ОПИСАНИЕ ДОПОЛНИТЕЛЬНЫХ НАСТРОЙЕК RT.WIDESTORE

##### distributed\_product\_mode

Изменяет поведение [распределенных подзапросов](https://clickhouse.tech/docs/ru/sql-reference/operators/).

RT.WideStore 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](https://clickhouse.tech/docs/ru/sql-reference/table-functions/remote/).

Возможные значения:

* deny – значение по умолчанию. Запрещает использование таких подзапросов (При попытке использование 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.

##### enable\_optimize\_predicate\_expression

Включает пробрасывание предикатов в подзапросы для запросов SELECT.

Пробрасывание 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 RT.WideStore 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](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/replication/).

Из устаревших реплик таблицы RT.WideStore 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 RT.WideStore 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](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/).

##### force\_primary\_key

Запрещает выполнение запросов, если использовать индекс по первичному ключу невозможно.

Работает с таблицами семейства MergeTree.

При force\_primary\_key=1 RT.WideStore 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](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/).

##### format\_schema

Параметр применяется в том случае, когда используются форматы, требующие определения схемы, например [Cap’n Proto](https://capnproto.org/) или [Protobuf](https://developers.google.com/protocol-buffers/). Значение параметра зависит от формата.

##### fsync\_metadata

Включает или отключает [fsync](https://pubs.opengroup.org/onlinepubs/9699919799/functions/fsync.html) при записи .sql файлов. По умолчанию включено.

Имеет смысл выключать, если на сервере миллионы мелких таблиц-чанков, которые постоянно создаются и уничтожаются.

##### enable\_http\_compression

Включает или отключает сжатие данных в ответе на HTTP-запрос.

Для получения дополнительной информации, читайте [Описание интерфейса HTTP](https://clickhouse.tech/docs/ru/interfaces/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](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-enable_http_compression).

Возможные значения: числа от 1 до 9.

Значение по умолчанию: 3.

##### http\_native\_compression\_disable\_checksumming\_on\_decompress

Включает или отключает проверку контрольной суммы при распаковке данных HTTP POST от клиента. Используется только для собственного (Navite) формата сжатия RT.WideStore (ни gzip, ни deflate).

Для получения дополнительной информации, читайте [Описание интерфейса HTTP](https://clickhouse.tech/docs/ru/interfaces/http/).

Возможные значения:

* 0 – выключена.
* 1 – включена.

Значение по умолчанию: 0.

##### 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](https://clickhouse.tech/docs/ru/interfaces/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](https://clickhouse.tech/docs/ru/engines/table-engines/special/url/) при выполнении HTTP запросов методом GET. Настройка применяется для обоих типов таблиц: созданных запросом [CREATE TABLE](https://clickhouse.tech/docs/ru/operations/sql_reference/create/#create-table-query) и с помощью табличной функции [url](https://clickhouse.tech/docs/ru/sql-reference/table-functions/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 RT.WideStore игнорирует строку и переходит к следующей.

В случае превышения input\_format\_allow\_errors\_num RT.WideStore 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 RT.WideStore игнорирует строку и переходит к следующей.

В случае превышения input\_format\_allow\_errors\_ratio RT.WideStore генерирует исключение.

##### input\_format\_values\_interpret\_expressions

Включает или отключает парсер SQL, если потоковый парсер не может проанализировать данные. Этот параметр используется только для формата [Values](https://clickhouse.tech/docs/ru/interfaces/formats/#data-format-values) при вставке данных. Дополнительные сведения о парсерах читайте в разделе [Синтаксис](https://clickhouse.tech/docs/ru/sql-reference/syntax/).

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, RT.WideStore ведет себя так, как будто значение параметра равно 0.

Значение по умолчанию: 1.

Пример использования:

Вставим значение типа [DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/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](https://clickhouse.tech/docs/ru/interfaces/formats/#data-format-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 интерпретируется 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, RT.WideStore 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, RT.WideStore проверяет фактический тип литерала и использует шаблон выражения соответствующего типа. В некоторых случаях это может значительно замедлить оценку выажения 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](https://clickhouse.tech/docs/ru/interfaces/formats/#jsoneachrow).
* [CSV](https://clickhouse.tech/docs/ru/interfaces/formats/#csv).
* [TabSeparated](https://clickhouse.tech/docs/ru/interfaces/formats/#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 │

└─────┴────────┘

┌──Id─┬─Value──┐

│ 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 использование значений по умолчанию в случаях, когда во входных данных содержится NULL, но тип соответствующего столбца не Nullable(T) (для текстовых форматов).

##### 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, RT.WideStore генерирует исключение. Если пропускание вставки включено, RT.WideStore не вставляет неизвестные данные и не генерирует исключение.

Поддерживаемые форматы:

* [JSONEachRow](https://clickhouse.tech/docs/ru/interfaces/formats/#jsoneachrow),
* [CSVWithNames](https://clickhouse.tech/docs/ru/interfaces/formats/#csvwithnames),
* [TabSeparatedWithNames](https://clickhouse.tech/docs/ru/interfaces/formats/#tabseparatedwithnames),
* [TSKV](https://clickhouse.tech/docs/ru/interfaces/formats/#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](https://clickhouse.tech/docs/ru/interfaces/formats/#jsoneachrow)

Возможные значения:

* 0 – выключена.
* 1 – включена.

Значение по умолчанию: 0.

См. также:

* [Использование вложенных структур](https://clickhouse.tech/docs/ru/interfaces/formats/#jsoneachrow-nested) 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](https://clickhouse.tech/docs/ru/interfaces/formats/#csvwithnames),
* [TabSeparatedWithNames](https://clickhouse.tech/docs/ru/interfaces/formats/#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](https://clickhouse.tech/docs/ru/sql-reference/functions/date-time-functions/).

Возможные значения:

* best\_effort – включает расширенный парсинг.

RT.WideStore может парсить базовый формат YYYY-MM-DD HH:MM:SS и все форматы [ISO 8601](https://en.wikipedia.org/wiki/ISO_8601). Например, 2018-06-08T01:02:03.000Z.

* basic – используется базовый парсер.

RT.WideStore 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.](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/)
* [Функции для работы с датой и временем.](https://clickhouse.tech/docs/ru/sql-reference/functions/date-time-functions/)

##### date\_time\_output\_format

Позволяет выбрать разные выходные форматы текстового представления даты и времени.

Возможные значения:

* simple – простой выходной формат.

Выходные дата и время RT.WideStore 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 RT.WideStore в формате [ISO 8601](https://en.wikipedia.org/wiki/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](https://en.wikipedia.org/wiki/Unix_time). Например: 1566285536.

Значение по умолчанию: simple.

См. также:

* [Тип данных DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/).
* [Функции для работы с датой и временем](https://clickhouse.tech/docs/ru/sql-reference/functions/date-time-functions/).

##### join\_default\_strictness

Устанавливает строгость по умолчанию для [JOIN](https://clickhouse.tech/docs/ru/sql-reference/statements/select/join/#select-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 RT.WideStore генерирует исключение.

Значение по умолчанию: ALL.

##### 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](https://clickhouse.tech/docs/ru/engines/table-engines/special/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](https://clickhouse.tech/docs/ru/sql-reference/statements/select/join/#select-join).
* [Движок таблиц Join](https://clickhouse.tech/docs/ru/engines/table-engines/special/join/).
* [join\_default\_strictness](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-join_default_strictness).

##### join\_use\_nulls

Устанавливает тип поведения [JOIN](https://clickhouse.tech/docs/ru/sql-reference/statements/select/join/). При объединении таблиц могут появиться пустые ячейки. RT.WideStore 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](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/#data_type-nullable), а пустые ячейки заполняются значениями [NULL](https://clickhouse.tech/docs/ru/sql-reference/syntax/).

##### partial\_merge\_join\_optimizations

Отключает все оптимизации для запросов [JOIN](https://clickhouse.tech/docs/ru/sql-reference/statements/select/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 RT.WideStore, 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](https://clickhouse.tech/docs/ru/sql-reference/statements/select/join/) с частичным MergeJoin алгоритмом.

Сервер RT.WideStore:

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

Устанавливет 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](https://ru.wikipedia.org/wiki/LZ4)
* NONE – не применять сжатие.

Значение по умолчанию: LZ4.

##### any\_join\_distinct\_right\_table\_keys

Включает устаревшее поведение сервера RT.WideStore 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 RT.WideStore 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 RT.WideStore 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](https://clickhouse.tech/docs/ru/sql-reference/statements/select/join/#join-settings).

##### max\_block\_size

Данные в RT.WideStore 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](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/) RT.WideStore 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](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/) превышает merge\_tree\_min\_rows\_for\_concurrent\_read, то RT.WideStore 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](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/), превышает значение merge\_tree\_min\_bytes\_for\_concurrent\_read, то RT.WideStore 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 RT.WideStore 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 RT.WideStore 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 RT.WideStore проверяет засечки данных в файле индекса. Если RT.WideStore 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, RT.WideStore не используют кэш несжатых блоков.

Кэш несжатых блоков хранит данные, извлечённые при выполнении запросов. RT.WideStore 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](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/#server-settings-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, RT.WideStore не используют кэш несжатых блоков.

Кэш несжатых блоков хранит данные, извлечённые при выполнении запросов. RT.WideStore использует кэш для ускорения ответов на повторяющиеся небольшие запросы. Настройка защищает кэш от переполнения. Настройка сервера [uncompressed\_cache\_size](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/#server-settings-uncompressed_cache_size) определяет размер кэша несжатых блоков.

Возможное значение:

* Положительное целое число.

Значение по умолчанию: 2013265920.

##### min\_bytes\_to\_use\_direct\_io

Минимальный объём данных, необходимый для прямого (небуферизованного) чтения/записи (direct I/O) на диск.

RT.WideStore 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 RT.WideStore 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](https://clickhouse.tech/docs/ru/interfaces/cli/).

Возможные значения:

* LZ4 – устанавливает метод сжатия LZ4.
* ZSTD – устанавливает метод сжатия ZSTD.

Значение по умолчанию: LZ4.

См. также:

* [network\_zstd\_compression\_level](https://clickhouse.tech/docs/ru/operations/settings/settings/#network_zstd_compression_level).

##### network\_zstd\_compression\_level

Регулирует уровень сжатия ZSTD. Используется только тогда, когда [network\_compression\_method](https://clickhouse.tech/docs/ru/operations/settings/settings/#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](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/#server_configuration_parameters-query-log).

Пример:

log\_queries=1

##### log\_queries\_min\_query\_duration\_ms

Минимальное время выполнения запроса для логгирования 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 (логгировать 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 RT.WideStore с этой установкой, записывается согласно правилам конфигурационного параметра сервера [query\_thread\_log](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/#server_configuration_parameters-query_thread_log).

Пример:

log\_query\_threads=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](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/replication/).

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.

Если на сервере обычно исполняется менее одного запроса SELECT одновременно, то выставите этот параметр в значение чуть меньше количества реальных процессорных ядер.

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](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-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 RT.WideStore.*

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](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/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 RT.WideStore.*

##### max\_query\_size

Максимальный кусок запроса, который будет считан в оперативку для разбора парсером языка SQL.

Запрос INSERT также содержит данные для INSERT-а, которые обрабатываются отдельным, потоковым парсером (расходующим 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 обрывает 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.

##### 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](https://clickhouse.tech/docs/ru/engines/table-engines/integrations/kafka/#kafka) перед повторной попыткой.

Возможные значения:

* Положительное целое число.
* 0 – бесконечный таймаут.

Значение по умолчанию: 5000.

См. также:

* [Apache Kafka](https://kafka.apache.org/).

##### 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](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/#server-settings-uncompressed_cache_size) (настраивается только в конфигурационном файле) – размер кэша разжатых блоков. По умолчанию - 8 GiB. Кэш разжатых блоков заполняется по мере надобности, 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 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](https://clickhouse.tech/docs/ru/operations/settings/settings/#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](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-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.

RT.WideStore поддерживает следующие алгоритмы выбора реплик:

* [Random](https://clickhouse.tech/docs/ru/operations/settings/settings/#load_balancing-random) (by default),
* [Nearest hostname](https://clickhouse.tech/docs/ru/operations/settings/settings/#load_balancing-nearest_hostname),
* [In order](https://clickhouse.tech/docs/ru/operations/settings/settings/#load_balancing-in_order),
* [First or random](https://clickhouse.tech/docs/ru/operations/settings/settings/#load_balancing-first_or_random),
* [Round robin](https://clickhouse.tech/docs/ru/operations/settings/settings/#load_balancing-round_robin).

См. также:

* [distributed\_replica\_max\_ignored\_errors](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-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.widestore.rt.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, предпочитается 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 обрабатываю 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 – RT.WideStore всегда отправляет запрос на localhost реплику, если она существует.
* 0 – RT.WideStore использует балансировку, заданную настройкой [load\_balancing](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-load_balancing).

Значение по умолчанию: 1.

Warning

Отключайте эту настройку при использовании [max\_parallel\_replicas](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-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.

Для консистентности (чтобы получить разные части одного и того же разбиения), эта опция работает только при заданном ключе сэмплирования.

Отставание реплик не контролируется.

##### compile

Включить компиляцию запросов. По умолчанию - 0 (выключено).

Компиляция предусмотрена только для части конвейера обработки запроса - для первой стадии агрегации (GROUP BY).

В случае, если эта часть конвейера была скомпилирована, запрос может работать быстрее, за счёт разворачивания коротких циклов и инлайнинга вызовов агрегатных функций. Максимальный прирост производительности (до четырёх раз в редких случаях) достигается на запросах с несколькими простыми агрегатными функциями. Как правило, прирост производительности незначителен. В очень редких случаях возможно замедление выполнения запроса.

##### min\_count\_to\_compile

После скольких раз, когда скомпилированный кусок кода мог пригодиться, выполнить его компиляцию. По умолчанию - 3.

Для тестирования можно установить значение 0: компиляция выполняется синхронно, и запрос ожидает окончания процесса компиляции перед продолжением выполнения. Во всех остальных случаях используйте значения, начинающиеся с 1. Как правило, компиляция занимает по времени около 5-10 секунд.

В случае, если значение равно 1 или больше, компиляция выполняется асинхронно, в отдельном потоке. При готовности результата, он сразу же будет использован, в том числе, уже выполняющимися в данный момент запросами.

Скомпилированный код требуется для каждого разного сочетания используемых в запросе агрегатных функций и вида ключей в GROUP BY.

Результаты компиляции сохраняются в директории build в виде .so файлов. Количество результатов компиляции не ограничено, так как они не занимают много места. При перезапуске сервера, старые результаты будут использованы, за исключением случая обновления сервера - тогда старые результаты удаляются.

##### 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

Если значение истинно, то при использовании JSON\* форматов UInt64 и Int64 числа выводятся в кавычках (из соображений совместимости с большинством реализаций JavaScript), иначе - без кавычек.

##### output\_format\_json\_quote\_denormals

При выводе данных в формате [JSON](https://clickhouse.tech/docs/ru/interfaces/formats/#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 RT.WideStore 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 RT.WideStore 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](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-select_sequential_consistency).

RT.WideStore 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](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-insert_quorum_timeout),
* [select\_sequential\_consistency](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-select_sequential_consistency).

##### insert\_quorum\_timeout

Время ожидания кворумной записи в миллисекундах. Если время прошло, а запись так не состоялась, то RT.WideStore 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](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-insert_quorum).
* [select\_sequential\_consistency](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-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 RT.WideStore 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 RT.WideStore сгенерирует исключение. В запросе SELECT не будут участвовать данные, которые ещё не были записаны на кворум реплик.

См. также:

* [insert\_quorum](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-insert_quorum).
* [insert\_quorum\_timeout](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-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](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/replication/)).

##### 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 …)](https://clickhouse.tech/docs/ru/sql-reference/aggregate-functions/reference/count/#agg_function-count).

Возможные значения:

* [uniq](https://clickhouse.tech/docs/ru/sql-reference/aggregate-functions/reference/uniq/#agg_function-uniq).
* [uniqCombined](https://clickhouse.tech/docs/ru/sql-reference/aggregate-functions/reference/uniqcombined/#agg_function-uniqcombined).
* [uniqCombined64](https://clickhouse.tech/docs/ru/sql-reference/aggregate-functions/reference/uniqcombined64/#agg_function-uniqcombined64).
* [uniqHLL12](https://clickhouse.tech/docs/ru/sql-reference/aggregate-functions/reference/uniqhll12/#agg_function-uniqhll12).
* [uniqExact](https://clickhouse.tech/docs/ru/sql-reference/aggregate-functions/reference/uniqexact/#agg_function-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 каждому отдельному 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 каждому отдельному 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:

* RT.WideStore не может установить соединение с репликой по любой причине.

RT.WideStore 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 разрешаться с помощью DNS и это не ошибка.
  + Ошибка конфигурации. Конфигурационный файл RT.WideStore может содержать неправильное имя хоста.

Возможные значения:

* 1 – пропуск включен.

Если шард недоступен, то RT.WideStore возвращает результат, основанный на неполных данных и не оповещает о проблемах с доступностью хостов.

* 0 – пропуск выключен.

Если шард недоступен, то RT.WideStore генерирует исключение.

Значение по умолчанию: 0.

##### optimize\_skip\_unused\_shards

Включает или отключает пропуск неиспользуемых шардов для запросов [SELECT](https://clickhouse.tech/docs/ru/sql-reference/statements/select/) , в которых условие ключа шардирования задано в секции WHERE/PREWHERE. Предполагается, что данные распределены с помощью ключа шардирования, в противном случае настройка ничего не делает.

Возможные значения:

* 0 – выключена.
* 1 – включена.

Значение по умолчанию: 0.

##### optimize\_skip\_unused\_shards\_nesting

Контролирует настройку [optimize\_skip\_unused\_shards](https://clickhouse.tech/docs/ru/operations/settings/settings/#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](https://clickhouse.tech/docs/ru/operations/settings/settings/#optimize-skip-unused-shards) включена, а пропуск неиспользуемых шардов невозможен. Если данная настройка включена и пропуск невозможен, RT.WideStore 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](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-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](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-force_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](https://clickhouse.tech/docs/ru/sql-reference/statements/misc/#misc_operations-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.

##### 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](https://clickhouse.tech/docs/ru/operations/settings/settings/#load_balancing-round_robin).
* [Table engine Distributed](https://clickhouse.tech/docs/ru/engines/table-engines/special/distributed/).
* [distributed\_replica\_error\_cap](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-distributed_replica_error_cap).
* [distributed\_replica\_max\_ignored\_errors](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-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](https://clickhouse.tech/docs/ru/operations/settings/settings/#load_balancing-round_robin).
* [Table engine Distributed](https://clickhouse.tech/docs/ru/engines/table-engines/special/distributed/).
* [distributed\_replica\_error\_half\_life](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-distributed_replica_error_half_life).
* [distributed\_replica\_max\_ignored\_errors](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-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](https://clickhouse.tech/docs/ru/operations/settings/settings/#load_balancing-round_robin).
* [Table engine Distributed](https://clickhouse.tech/docs/ru/engines/table-engines/special/distributed/).
* [distributed\_replica\_error\_cap](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-distributed_replica_error_cap).
* [distributed\_replica\_error\_half\_life](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-distributed_replica_error_half_life).

##### distributed\_directory\_monitor\_sleep\_time\_ms

Основной интервал отправки данных движком таблиц [Distributed](https://clickhouse.tech/docs/ru/engines/table-engines/special/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](https://clickhouse.tech/docs/ru/engines/table-engines/special/distributed/). Ограничивает экпоненциальный рост интервала, установленого настройкой [distributed\_directory\_monitor\_sleep\_time\_ms](https://clickhouse.tech/docs/ru/operations/settings/settings/#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](https://clickhouse.tech/docs/ru/engines/table-engines/special/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](https://en.wikipedia.org/wiki/Nice_(Unix))) 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 бысто 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](https://clickhouse.tech/docs/ru/operations/optimizing-performance/sampling-query-profiler/). 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](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Значение по умолчанию: 1000000000 наносекунд (раз в секунду).

См. также:

* Системная таблица [trace\_log](https://clickhouse.tech/docs/ru/operations/system-tables/trace_log/#system_tables-trace_log).

##### query\_profiler\_cpu\_time\_period\_ns

Устанавливает период для таймера CPU [query profiler](https://clickhouse.tech/docs/ru/operations/optimizing-performance/sampling-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](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Значение по умолчанию: 1000000000 наносекунд.

См. также:

* Системная таблица [trace\_log](https://clickhouse.tech/docs/ru/operations/system-tables/trace_log/#system_tables-trace_log).

##### allow\_introspection\_functions

Включает или отключает [функции самоанализа](https://clickhouse.tech/docs/ru/sql-reference/functions/introspection/) для профилирования запросов.

Возможные значения:

* 1 – включены функции самоанализа.
* 0 – функции самоанализа отключены.

Значение по умолчанию: 0.

См. Также:

* [Sampling Query Profiler](https://clickhouse.tech/docs/ru/operations/optimizing-performance/sampling-query-profiler/).
* Системная таблица [trace\_log](https://clickhouse.tech/docs/ru/operations/system-tables/trace_log/#system_tables-trace_log).

##### input\_format\_parallel\_parsing

* Тип: bool
* Значение по умолчанию: True.

Включает режим, при котором входящие данные парсятся параллельно, но с сохранением исходного порядка следования. Поддерживается только для форматов TSV, TKSV, CSV и JSONEachRow.

##### 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](https://google.github.io/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.

озможные 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 c движком [MergeTree](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/)). Настройка применяется при запуске сервера RT.WideStore и не может быть изменена во пользовательском сеансе. Настройка позволяет управлять загрузкой процессора и диска. Чем меньше пулл, тем ниже нагрузка на CPU и диск, при этом фоновые процессы замедляются, что может повлиять на скорость выполнения запроса.

Допустимые значения:

* Положительное целое число.

Значение по умолчанию: 16.

##### 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](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/replication/) 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](https://clickhouse.tech/docs/ru/engines/table-engines/special/distributed/#distributed)).

По умолчанию RT.WideStore вставляет данные в распределённую таблицу в асинхронном режиме. Если insert\_distributed\_sync=1, то данные вставляются сихронно, 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](https://clickhouse.tech/docs/ru/engines/table-engines/special/distributed/#distributed).
* [Управление распределёнными таблицами](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query-language-system-distributed).

##### validate\_polygons

Включает или отключает генерирование исключения в функции [pointInPolygon](https://clickhouse.tech/docs/ru/sql-reference/functions/geo/#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](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/replication/).

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](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/replication/).

##### transform\_null\_in

Разрешает сравнивать значения [NULL](https://clickhouse.tech/docs/ru/sql-reference/syntax/#null-literal) в операторе [IN](https://clickhouse.tech/docs/ru/sql-reference/operators/in/).

По умолчанию, значения NULL нельзя сравнивать, поскольку NULL обозначает неопределённое 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 │

└──────┴───────┘

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](https://clickhouse.tech/docs/ru/sql-reference/operators/in/#in-null-processing).

##### 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, RT.WideStore 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 RT.WideStore 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](https://clickhouse.tech/docs/ru/interfaces/formats/#native).

Если использование типа LowCardinality ограничено, сервер RT.WideStore 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 RT.WideStore 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](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/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](https://clickhouse.tech/docs/ru/engines/table-engines/special/buffer/). Настройка применяется при запуске сервера RT.WideStore 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](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/#table_engine-mergetree-multiple-volumes). Настройка применяется при запуске сервера RT.WideStore 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](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/replication/) таблиц, стримов в [Kafka](https://clickhouse.tech/docs/ru/engines/table-engines/integrations/kafka/) и обновления IP адресов у записей во внутреннем [DNS кеше](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/#server-settings-dns-cache-update-period). Настройка применяется при запуске сервера RT.WideStore и не может быть изменена в пользовательском сеансе.

Допустимые значения:

* Положительное целое число.

Значение по умолчанию: 16.

##### background\_distributed\_schedule\_pool\_size

Задает количество потоков для выполнения фоновых задач. Работает для таблиц с движком [Distributed](https://clickhouse.tech/docs/ru/engines/table-engines/special/distributed/). Настройка применяется при запуске сервера RT.WideStore 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 RT.WideStore и не может быть изменена в пользовательском сеансе.

Допустимые значения:

* Положительное целое число.

Значение по умолчанию: 16.

Смотрите также

* Движок [Kafka](https://clickhouse.tech/docs/ru/engines/table-engines/integrations/kafka/#kafka).
* Движок [RabbitMQ](https://clickhouse.tech/docs/ru/engines/table-engines/integrations/rabbitmq/#rabbitmq-engine).

##### format\_avro\_schema\_registry\_url

Задает URL реестра схем [Confluent](https://docs.confluent.io/current/schema-registry/index.html) для использования с форматом [AvroConfluent](https://clickhouse.tech/docs/ru/interfaces/formats/#data-format-avro-confluent).

Значение по умолчанию: Пустая строка.

##### input\_format\_avro\_allow\_missing\_fields

Позволяет использовать данные, которых не нашлось в схеме формата [Avro](https://clickhouse.tech/docs/ru/interfaces/formats/#data-format-avro) или [AvroConfluent](https://clickhouse.tech/docs/ru/interfaces/formats/#data-format-avro-confluent). Если поле не найдено в схеме, RT.WideStore 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](https://clickhouse.tech/docs/ru/sql-reference/statements/create/view/#create-view). 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](https://clickhouse.tech/docs/ru/operations/settings/settings/#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](https://clickhouse.tech/docs/ru/sql-reference/statements/create/view/#create-view). 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](https://clickhouse.tech/docs/ru/operations/settings/settings/#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;

+-a-+

| 1 |

+---+

##### optimize\_read\_in\_order

Включает или отключает оптимизацию в запросах [SELECT](https://clickhouse.tech/docs/ru/sql-reference/statements/select/) с секцией [ORDER BY](https://clickhouse.tech/docs/ru/sql-reference/statements/select/order-by/#optimize_read_in_order) при работе с таблицами семейства [MergeTree](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/).

Возможные значения:

* 0 – оптимизация отключена.
* 1 – оптимизация включена.

Значение по умолчанию: 1.

См. также:

* [Оптимизация чтения данных](https://clickhouse.tech/docs/ru/sql-reference/statements/select/order-by/#optimize_read_in_order) в секции ORDER BY.

##### optimize\_aggregation\_in\_order

Включает или отключает оптимизацию в запросах [SELECT](https://clickhouse.tech/docs/ru/sql-reference/statements/select/) с секцией [GROUP BY](https://clickhouse.tech/docs/ru/sql-reference/statements/select/group-by/) при наличии подходящих ключей сортировки. Используется при работе с таблицами [MergeTree](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/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](https://clickhouse.tech/docs/ru/sql-reference/statements/select/group-by/#aggregation-in-order).

##### mutations\_sync

Позволяет выполнять запросы ALTER TABLE ... UPDATE|DELETE ([мутации](https://clickhouse.tech/docs/ru/sql-reference/statements/alter/#mutations)) 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](https://clickhouse.tech/docs/ru/sql-reference/statements/alter/#synchronicity-of-alter-queries).
* [Мутации](https://clickhouse.tech/docs/ru/sql-reference/statements/alter/#mutations).

##### ttl\_only\_drop\_parts

Для таблиц [MergeTree](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/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](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/#mergetree-query-clauses) (настройка merge\_with\_ttl\_timeout)
* [Table TTL](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/#mergetree-table-ttl).

##### output\_format\_pretty\_max\_value\_width

Ограничивает длину значения, выводимого в формате [Pretty](https://clickhouse.tech/docs/ru/interfaces/formats/#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:

┌─range(number)─┐

│ ⋯ │

│ ⋯ │

│ ⋯ │

│ ⋯ │

│ ⋯ │

└───────────────┘

##### output\_format\_pretty\_row\_numbers

Включает режим отображения номеров строк для запросов, выводимых в формате [Pretty](https://clickhouse.tech/docs/ru/interfaces/formats/#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](https://clickhouse.tech/docs/ru/operations/system-tables/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. │

└──────────────────────────┴───────┴───────────────────────────────────────────────────────┘

##### allow\_experimental\_bigint\_types

Включает или отключает поддержку целочисленных значений, превышающих максимальное значение, допустимое для типа int.

Возможные значения:

* 1 – большие целочисленные значения поддерживаются.
* 0 – большие целочисленные значения не поддерживаются.

Значение по умолчанию: 0.

##### 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](https://clickhouse.tech/docs/ru/sql-reference/functions/type-conversion-functions/#type_conversion_function-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](https://clickhouse.tech/docs/ru/sql-reference/functions/type-conversion-functions/#type_conversion_function-cast).

##### persistent

Отключает перманентность для табличных движков [Set](https://clickhouse.tech/docs/ru/engines/table-engines/special/set/#set) и [Join](https://clickhouse.tech/docs/ru/engines/table-engines/special/join/#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\_tsv\_null\_representation

Определяет представление NULL для формата выходных данных [TSV](https://clickhouse.tech/docs/ru/interfaces/formats/#tabseparated). 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](https://clickhouse.tech/docs/ru/interfaces/formats/#jsoneachrow).

Возможные значения:

* 1 – RT.WideStore выводит все строки в виде массива и при этом каждую строку в формате JSONEachRow.
* 0 – RT.WideStore 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](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/#data_type-nullable) для ключей таблиц [MergeTree](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/#table_engines-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](https://clickhouse.tech/docs/ru/sql-reference/aggregate-functions/combinators/#agg-functions-combinator-ornull). Включите для совместимости со стандартом SQL.  
Реализуется с помощью перезаписи запросов (аналогично настройке [count\_distinct\_implementation](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-count_distinct_implementation)), 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:

┌─SUM(-1)─┬─MAX(0)─┐

│ 0 │ 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](https://clickhouse.tech/docs/ru/sql-reference/statements/select/union/) без явного указания UNION ALL или UNION DISTINCT.

Возможные значения:

* 'DISTINCT' – RT.WideStore выводит строки в результате объединения результатов запросов, удаляя повторяющиеся строки.
* 'ALL' – RT.WideStore выводит все строки в результате объединения результатов запросов, включая повторяющиеся строки.
* '' – RT.WideStore генерирует исключение при использовании с UNION.

Значение по умолчанию: ''.

Смотрите примеры в разделе [UNION](https://clickhouse.tech/docs/ru/sql-reference/statements/select/union/).

##### data\_type\_default\_nullable

Позволяет использовать по умолчанию тип данных [Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/#data_type-nullable) в определении столбца без явных модификаторов [NULL или NOT NULL](https://clickhouse.tech/docs/ru/sql-reference/statements/create/table/#null-modifiers).

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](https://clickhouse.tech/docs/ru/sql-reference/statements/select/from/#select-from-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 RT.WideStore начнет трассировку для выполненных запросов (если не указан [входящий контекст](https://www.w3.org/TR/trace-context/) 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, RT.WideStore 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](https://clickhouse.tech/docs/ru/sql-reference/statements/create/view/#materialized) и БД [MaterializeMySQL](https://clickhouse.tech/docs/ru/engines/database-engines/materialize-mysql/).

##### allow\_experimental\_geo\_types

Разрешает использование экспериментальных типов данных для работы с [географическими структурами](https://clickhouse.tech/docs/ru/sql-reference/data-types/geo/).

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.