**RT.WIDESTORE**

**РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

**ОГЛАВЛЕНИЕ**

[1 ВВЕДЕНИЕ 5](#_Toc75527502)

[1.1 Наименование и условное обозначение 5](#_Toc75527503)

[1.2 Область применения 5](#_Toc75527504)

[1.3 Термины, определения и сокращения 5](#_Toc75527505)

[1.4 Краткое описание возможностей 6](#_Toc75527506)

[1.5 Требования к квалификации 6](#_Toc75527507)

[2 Подготовка к работе 6](#_Toc75527508)

[2.1 Требования к доступам 6](#_Toc75527509)

[2.2 Порядок проверки работоспособности 6](#_Toc75527510)

[3 Работа с СУБД RT.WideStore 7](#_Toc75527511)

[3.1 Подключение к СУБД RT.WideStore 7](#_Toc75527512)

[3.1.1 Клиент командной строки 7](#_Toc75527513)

[3.1.1.1 Использование 7](#_Toc75527514)

[3.1.1.2 Запросы с параметрами 9](#_Toc75527515)

[3.1.1.3 Конфигурирование 10](#_Toc75527516)

[3.1.1.3.1 Параметры командной строки 10](#_Toc75527517)

[3.1.1.3.2 Конфигурационные файлы 11](#_Toc75527518)

[3.1.2 Родной интерфейс (TCP) 11](#_Toc75527519)

[3.1.3 HTTP-интерфейс 11](#_Toc75527520)

[3.1.3.1 Сжатие 13](#_Toc75527521)

[3.1.3.2 База данных по умолчанию 14](#_Toc75527522)

[3.1.3.3 Буферизация ответа 16](#_Toc75527523)

[3.1.3.4 Запросы с параметрами 16](#_Toc75527524)

[3.1.3.5 Предопределенный HTTP интерфейс 16](#_Toc75527525)

[3.1.3.5.1 predefined\_query\_handler 19](#_Toc75527526)

[3.1.3.5.2 dynamic\_query\_handler 19](#_Toc75527527)

[3.1.3.5.3 static 20](#_Toc75527528)

[3.1.4 MySQL-интерфейс 22](#_Toc75527529)

[3.2 Движки таблиц 23](#_Toc75527530)

[3.2.1 Семейства движков 23](#_Toc75527531)

[3.2.1.1 MergeTree 23](#_Toc75527532)

[3.2.1.2 Log 23](#_Toc75527533)

[3.2.1.3 Движки для интеграции 24](#_Toc75527534)

[3.2.1.4 Специальные движки 24](#_Toc75527535)

[3.2.2 Виртуальные столбцы 24](#_Toc75527536)

[3.3 Справка по SQL в RT.WideStore 25](#_Toc75527537)

[3.3.1 SQL выражения в RT.WideStore 25](#_Toc75527538)

[3.3.2 Синтаксис 25](#_Toc75527539)

[3.3.2.1 Пробелы 26](#_Toc75527540)

[3.3.2.2 Комментарии 26](#_Toc75527541)

[3.3.2.3 Ключевые слова 26](#_Toc75527542)

[3.3.2.4 Идентификаторы 26](#_Toc75527543)

[3.3.2.5 Литералы 27](#_Toc75527544)

[3.3.2.5.1 Числовые 27](#_Toc75527545)

[3.3.2.5.2 Строковые 27](#_Toc75527546)

[3.3.2.5.3 Составные 27](#_Toc75527547)

[3.3.2.5.4 NULL 28](#_Toc75527548)

[3.3.2.6 Функции, синтаксис 28](#_Toc75527549)

[3.3.2.7 Операторы 28](#_Toc75527550)

[3.3.2.8 Типы данных и движки таблиц 28](#_Toc75527551)

[3.3.2.9 Синонимы выражений 28](#_Toc75527552)

[3.3.2.9.1 Примечания по использованию 29](#_Toc75527553)

[3.3.2.10 Звёздочка 29](#_Toc75527554)

[3.3.2.11 Выражения 30](#_Toc75527555)

[3.3.3 Распределенные DDL запросы (секция ON CLUSTER) 30](#_Toc75527556)

[3.3.4 Функции 30](#_Toc75527557)

[3.3.4.1 Строгая типизация 30](#_Toc75527558)

[3.3.4.2 Склейка одинаковых выражений 31](#_Toc75527559)

[3.3.4.3 Типы результата 31](#_Toc75527560)

[3.3.4.4 Константы 31](#_Toc75527561)

[3.3.4.5 Неизменяемость 31](#_Toc75527562)

[3.3.4.6 Функции высшего порядка, оператор -> и функция lambda(params, expr) 31](#_Toc75527563)

[3.3.4.7 Обработка ошибок 32](#_Toc75527564)

[3.3.4.8 Вычисление выражений-аргументов 32](#_Toc75527565)

[3.3.4.9 Выполнение функций при распределённой обработке запроса 32](#_Toc75527566)

[ПРИЛОЖЕНИЕ 1. ОПИСАНИЕ SQL ВЫРАЖЕНИЙ В RT.WIDESTORE 34](#_Toc75527567)

[ПРИЛОЖЕНИЕ 2. ОПИСАНИЕ ФУНКЦИЙ В RT.WIDESTORE 71](#_Toc75527568)

# 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 |
| --- | --- |
| 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. |
| 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). |
| БД | База данных |
| ОС | Операционная система |
| Система | Система «RT.WideStore» |
| СУБД | Система управлением базами данных |
| HTTP | (англ. HyperText Transfer Protocol – «протокол передачи гипертекста») – протокол прикладного уровня передачи данных, изначально – в виде гипертекстовых документов в формате HTML, в настоящее время используется для передачи произвольных данных. |
| TCP | (англ. Transmission Control Protocol – протокол управления передачей) – один из основных протоколов передачи данных интернета, предназначенный для управления передачей данных интернета. Пакеты в TCP называются сегментами. |
| MySQL | Свободная реляционная система управления базами данных. Разработку и поддержку MySQL осуществляет корпорация Oracle, получившая права на торговую марку вместе с поглощённой Sun Microsystems, которая ранее приобрела шведскую компанию MySQL AB. |
| MySQL CLI | Консольный клиент MySQL для работы с базами данных. |
| URL | (от англ. Uniform Resource Locator – Унифицированный указатель ресурса) – система унифицированных адресов электронных ресурсов, или единообразный определитель местонахождения ресурса (файла). Используется как стандарт записи ссылок на объекты в Интернете. |
| Re2 | Библиотека регулярных выражений. |

## Краткое описание возможностей

СУБД RT.WideStore из-за своей архитектуры оптимальна для выполнения онлайн запросов используемых для построения отчетов на основе больших массивов данных, структурированных по многомерному принципу.

## Требования к квалификации

Пользователи Системы должен иметь навыки работы со следующими продуктами:

* Знание языка запросов (SQL);
* Умение работать с большим объёмом информации;
* Знание ОС, на которые установлена Система;
* Знание интерфейсов, на котором планируется взаимодействие с Системой;
* Формирование OLAP отчетов, в том числе для конечного пользователя.

# Подготовка к работе

## Требования к доступам

Администратор должен обладать необходимыми и достаточными доступами, в том числе сетевыми, к предустановленным в рамках Системы компонентам.

Для получения паролей к учетным записям для проведения работ на системе администратору необходимо обращаться в подразделение, осуществляющее поддержку соответствующих систем (RT.WideStore).

## Порядок проверки работоспособности

Первичная проверка работоспособности компонентов системы заключается в контроле получения доступа к схемам СУБД RT.WideStore, в том числе тестовым, где установлены компоненты Системы.

# Работа с СУБД RT.WideStore

Для выполнения работы с СУБД RT.WideStore необходимо рассмотреть следующие разделы:

* подключение к СУБД RT.WideStore,
* движки таблиц,
* справка по SQL в RT.WideStore.

## Подключение к СУБД RT.WideStore

Возможные подключения к СУБД RT.WideStore описаны в документе «RT.WideStore. Общее описание системы» в разделе 5.18 «Интерфейсы взаимодействия с Системой».

Более подробно разберем следующие варианты подключения:

* клиент командной строки,
* родной интерфейс (TCP),
* HTTP-интерфейс,
* MySQL-интерфейс.

### Клиент командной строки

RT.WideStore предоставляет собственный клиент командной строки: clickhouse-client. Клиент поддерживает запуск с аргументами командной строки и с конфигурационными файлами. Подробнее читайте в разделе [Конфигурирование](https://clickhouse.tech/docs/ru/interfaces/cli/#interfaces_cli_configuration).

Клиент [устанавливается](https://clickhouse.tech/docs/ru/getting-started/) пакетом clickhouse-client и запускается командой clickhouse-client.

$ clickhouse-client

ClickHouse client version 20.13.1.5273 (official build).

Connecting to localhost:9000 as user default.

Connected to ClickHouse server version 20.13.1 revision 54442.

:)

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

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

### Использование

Клиент может быть использован в интерактивном и не интерактивном (batch) режиме.

Чтобы использовать batch режим, укажите параметр query, или отправьте данные в stdin (проверяется, что stdin - не терминал), или и то, и другое.

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

Примеры использования клиента для вставки данных:

$ echo -ne "1, 'some text', '2016-08-14 00:00:00'\n2, 'some more text', '2016-08-14 00:00:01'" | clickhouse-client --database=test --query="INSERT INTO test FORMAT CSV";

$ cat <<\_EOF | clickhouse-client --database=test --query="INSERT INTO test FORMAT CSV";

3, 'some text', '2016-08-14 00:00:00'

4, 'some more text', '2016-08-14 00:00:01'

\_EOF

$ cat file.csv | clickhouse-client --database=test --query="INSERT INTO test FORMAT CSV";

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

По умолчанию, в batch режиме вы можете выполнить только один запрос. Чтобы выполнить несколько запросов из «скрипта», используйте параметр –multiquery. Это работает для всех запросов кроме INSERT. Результаты запросов выводятся подряд без дополнительных разделителей.

Также, при необходимости выполнить много запросов, вы можете запускать clickhouse-client на каждый запрос. Заметим, что запуск программы clickhouse-client может занимать десятки миллисекунд.

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

Если не указано multiline (по умолчанию):

* Чтобы выполнить запрос, нажмите Enter. Точка с запятой на конце запроса не обязательна. Чтобы ввести запрос, состоящий из нескольких строк, перед переводом строки, введите символ обратного слеша: \ - тогда после нажатия Enter, вам предложат ввести следующую строку запроса.

Если указано multiline (многострочный режим):

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

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

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

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

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

По умолчанию, в качестве формата, используется формат PrettyCompact (красивые таблички). Вы можете изменить формат с помощью секции FORMAT запроса, или с помощью указания \G на конце запроса, с помощью аргумента командной строки --format или --vertical, или с помощью конфигурационного файла клиента.

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

При выполнении запроса, клиент показывает:

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

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

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

### Запросы с параметрами

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

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

Cинтаксис запроса:

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

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

Где:

* name – идентификатор подстановки. В консольном клиенте его следует использовать как часть имени параметра --param\_<name> = value.
* data type – [тип данных](https://clickhouse.tech/docs/ru/sql-reference/data-types/) значения. Например, структура данных (integer, ('string', integer)) может иметь тип данных Tuple(UInt8, Tuple(String, UInt8)) ([целочисленный](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/) тип может быть и другим). В качестве параметра можно передать название столбца, таблицы и базы данных, в этом случае используется тип данных Identifier.

Пример:

$ clickhouse-client --param\_tuple\_in\_tuple="(10, ('dt', 10))" -q "SELECT \* FROM table WHERE val = {tuple\_in\_tuple:Tuple(UInt8, Tuple(String, UInt8))}"

$ clickhouse-client --param\_tbl="numbers" --param\_db="system" --param\_col="number" --query "SELECT {col:Identifier} FROM {db:Identifier}.{tbl:Identifier} LIMIT 10"

### Конфигурирование

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

* Командной строки.

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

* Конфигурационных файлов.

Параметры в конфигурационных файлах переопределяют значения по умолчанию.

### Параметры командной строки

Параметры командной строки в clickhouse-client:

* --host, -h – имя сервера, по умолчанию – ‘localhost’. Вы можете использовать как имя, так и IPv4 или IPv6 адрес.
* --port – порт для подключения, по умолчанию – 9000. Обратите внимание: для HTTP-интерфейса и нативного интерфейса используются разные порты.
* --user, -u – имя пользователя, по умолчанию – ‘default’.
* --password – пароль, по умолчанию – пустая строка.
* --query, -q – запрос для выполнения, при использовании в неинтерактивном режиме.
* --database, -d – выбрать текущую БД. Без указания значение берется из настроек сервера (по умолчанию – БД ‘default’).
* --multiline, -m – если указано – разрешить многострочные запросы, не отправлять запрос по нажатию Enter.
* --multiquery, -n – если указано – разрешить выполнять несколько запросов, разделённых точкой с запятой.
* --format, -f – использовать указанный формат по умолчанию для вывода результата.
* --vertical, -E – если указано, использовать по умолчанию формат [Vertical](https://clickhouse.tech/docs/ru/interfaces/formats/#vertical) для вывода результата. То же самое, что –format=Vertical. В этом формате каждое значение выводится на отдельной строке, что удобно для отображения широких таблиц.
* --time, -t – если указано, в неинтерактивном режиме вывести время выполнения запроса в поток ‘stderr’.
* --stacktrace – если указано, в случае исключения, выводить также его стек-трейс.
* --config-file – имя конфигурационного файла.
* --secure – если указано, будет использован безопасный канал.
* --param\_<name> – значение параметра для [запроса с параметрами](https://clickhouse.tech/docs/ru/interfaces/cli/#cli-queries-with-parameters).

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

### Конфигурационные файлы

clickhouse –client использует первый существующий файл из:

* определенного параметром --config-file,
* ./clickhouse-client.xml,
* ~/.clickhouse-client/config.xml,
* /etc/clickhouse-client/config.xml,

Пример конфигурационного файла:

<config>

<user>username</user>

<password>password</password>

<secure>False</secure>

</config>

### Родной интерфейс (TCP)

Нативный протокол используется в [клиенте командной строки](https://clickhouse.tech/docs/ru/interfaces/cli/), для взаимодействия между серверами во время обработки распределенных запросов, а также в других программах на C++. К сожалению, у родного протокола RT.WideStore пока нет формальной спецификации, но в нем можно разобраться с использованием исходного кода RT.WideStore и/или путем перехвата и анализа TCP трафика.

### HTTP-интерфейс

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

По умолчанию, clickhouse-server слушает HTTP на порту 8123 (это можно изменить в конфиге).

Если запросить GET / без параметров, то вернётся строка заданная с помощью настройки [http\_server\_default\_response](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/#server_configuration_parameters-http_server_default_response). Значение по умолчанию «Ok.» (с переводом строки на конце).

$ curl 'http://localhost:8123/'

Ok.

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

$ curl 'http://localhost:8123/ping'

Ok.

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

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

В случае успеха, вам вернётся код ответа 200 и результат обработки запроса в теле ответа.

В случае ошибки, вам вернётся код ответа 500 и текст с описанием ошибки в теле ответа.

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

Примеры:

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

1

$ wget -nv -O- 'http://localhost:8123/?query=SELECT 1'

1

$ echo -ne 'GET /?query=SELECT%201 HTTP/1.0\r\n\r\n' | nc localhost 8123

HTTP/1.0 200 OK

Date: Wed, 27 Nov 2019 10:30:18 GMT

Connection: Close

Content-Type: text/tab-separated-values; charset=UTF-8

X-ClickHouse-Server-Display-Name: clickhouse.ru-central1.internal

X-ClickHouse-Query-Id: 5abe861c-239c-467f-b955-8a201abb8b7f

X-ClickHouse-Summary: {"read\_rows":"0","read\_bytes":"0","written\_rows":"0","written\_bytes":"0","total\_rows\_to\_read":"0"}

1

Как видно, curl немного неудобен тем, что надо URL-эскейпить пробелы.  
Хотя wget сам всё эскейпит, но его не рекомендуется использовать, так как он плохо работает по HTTP 1.1 при использовании keep-alive и Transfer-Encoding: chunked.

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

1

$ echo 'SELECT 1' | curl 'http://localhost:8123/?query=' --data-binary @-

1

$ echo '1' | curl 'http://localhost:8123/?query=SELECT' --data-binary @-

1

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

Пример (так работать не будет):

$ echo 'ECT 1' | curl 'http://localhost:8123/?query=SEL' --data-binary @-

Code: 59, e.displayText() = DB::Exception: Syntax error: failed at position 0: SEL

ECT 1

, expected One of: SHOW TABLES, SHOW DATABASES, SELECT, INSERT, CREATE, ATTACH, RENAME, DROP, DETACH, USE, SET, OPTIMIZE., e.what() = DB::Exception

По умолчанию, данные возвращаются в формате TabSeparated.

Можно попросить любой другой формат - с помощью секции FORMAT запроса.

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

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

┏━━━┓

┃ 1 ┃

┡━━━┩

│ 1 │

└───┘

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

Примеры:

Создаём таблицу:

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

Используем привычный запрос INSERT для вставки данных:

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

Данные можно отправить отдельно от запроса:

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

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

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

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

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

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

$ curl 'http://localhost:8123/?query=SELECT%20a%20FROM%20t'

7

8

9

10

11

12

1

2

3

4

5

6

Удаляем таблицу:

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

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

### Сжатие

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

Вы можете использовать внутренний формат сжатия RT.WideStore при передаче данных. Формат сжатых данных нестандартный, и вам придётся использовать для работы с ним специальную программу clickhouse-compressor. Она устанавливается вместе с пакетом clickhouse-client. Для повышения эффективности вставки данных можно отключить проверку контрольной суммы на стороне сервера с помощью настройки [http\_native\_compression\_disable\_checksumming\_on\_decompress](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-http_native_compression_disable_checksumming_on_decompress).

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

Также можно использовать [сжатие HTTP](https://en.wikipedia.org/wiki/HTTP_compression). RT.WideStore поддерживает следующие [методы сжатия](https://en.wikipedia.org/wiki/HTTP_compression#Content-Encoding_tokens):

* gzip,
* br,
* deflate,
* xz.

Для отправки сжатого запроса POST, добавьте заголовок Content-Encoding: compression\_method.

Чтобы RT.WideStore сжимал ответ, разрешите сжатие настройкой [enable\_http\_compression](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-enable_http_compression) и добавьте заголовок Accept-Encoding: compression\_method. Уровень сжатия данных для всех методов сжатия можно задать с помощью настройки [http\_zlib\_compression\_level](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-http_zlib_compression_level).

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

Примеры:

# Отправка сжатых данных на сервер

$ echo "SELECT 1" | gzip -c | \

curl -sS --data-binary @- -H 'Content-Encoding: gzip' 'http://localhost:8123/'

# Получение сжатых данных с сервера

$ curl -vsS "http://localhost:8123/?enable\_http\_compression=1" \

-H 'Accept-Encoding: gzip' --output result.gz -d 'SELECT number FROM system.numbers LIMIT 3'

$ zcat result.gz

0

1

2

### База данных по умолчанию

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

$ echo 'SELECT number FROM numbers LIMIT 10' | curl 'http://localhost:8123/?database=system' --data-binary @-

0

1

2

3

4

5

6

7

8

9

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

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

1. С использованием HTTP Basic Authentication. Пример:

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

1. В параметрах URL user и password. Пример:

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

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

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

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

Также в параметрах URL вы можете указать любые настройки, которые будут использованы для обработки одного запроса, или целые профили настроек. Пример:<http://localhost:8123/?profile=web&max_rows_to_read=1000000000&query=SELECT+1>

Подробнее смотрите в документе: «RT.WideStore. Руководство администратора» п.3.10 «[Настройки](https://clickhouse.tech/docs/ru/operations/settings/)».

$ echo 'SELECT number FROM system.numbers LIMIT 10' | curl 'http://localhost:8123/?' --data-binary @-

0

1

2

3

4

5

6

7

8

9

Об остальных параметрах смотри раздел «SET».

Аналогично можно использовать RT.WideStore-сессии в HTTP-протоколе. Для этого необходимо добавить к запросу GET параметр session\_id. В качестве идентификатора сессии можно использовать произвольную строку. По умолчанию через 60 секунд бездействия сессия будет прервана. Можно изменить этот таймаут, изменяя настройку default\_session\_timeout в конфигурации сервера, или добавив к запросу GET параметр session\_timeout. Статус сессии можно проверить с помощью параметра session\_check=1. В рамках одной сессии одновременно может исполняться только один запрос.

Прогресс выполнения запроса можно отслеживать с помощью заголовков ответа X-ClickHouse-Progress. Для этого включите [send\_progress\_in\_http\_headers](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-send_progress_in_http_headers). Пример последовательности заголовков:

X-ClickHouse-Progress: {"read\_rows":"2752512","read\_bytes":"240570816","total\_rows\_to\_read":"8880128"}

X-ClickHouse-Progress: {"read\_rows":"5439488","read\_bytes":"482285394","total\_rows\_to\_read":"8880128"}

X-ClickHouse-Progress: {"read\_rows":"8783786","read\_bytes":"819092887","total\_rows\_to\_read":"8880128"}

Возможные поля заголовка:

* read\_rows – количество прочитанных строк.
* read\_bytes – объём прочитанных данных в байтах.
* total\_rows\_to\_read – общее количество строк для чтения.
* written\_rows – количество записанных строк.
* written\_bytes – объём записанных данных в байтах.

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

Может быть передан необязательный параметр query\_id - идентификатор запроса, произвольная строка. Подробнее смотрите раздел «Настройки, replace\_running\_query».

Может быть передан необязательный параметр quota\_key - ключ квоты, произвольная строка. Подробнее смотрите раздел «Квоты».

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

### Буферизация ответа

Существует возможность включить буферизацию ответа на стороне сервера. Для этого предусмотрены параметры URL buffer\_size и wait\_end\_of\_query.

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

Чтобы гарантировать буферизацию всего ответа необходимо выставить wait\_end\_of\_query=1. В этом случае данные, не поместившиеся в памяти, будут буферизованы во временном файле сервера.

Пример:

$ curl -sS 'http://localhost:8123/?max\_result\_bytes=4000000&buffer\_size=3000000&wait\_end\_of\_query=1' -d 'SELECT toUInt8(number) FROM system.numbers LIMIT 9000000 FORMAT RowBinary'

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

### Запросы с параметрами

Можно создать запрос с параметрами и передать для них значения из соответствующих параметров HTTP-запроса. Дополнительную информацию смотрите в [Запросы с параметрами для консольного клиента](https://clickhouse.tech/docs/ru/interfaces/cli/#cli-queries-with-parameters).

Пример:

$ curl -sS "http://localhost:8123/?param\_id=2&param\_phrase=test" -d "SELECT \* FROM table WHERE int\_column = {id:UInt8} and string\_column = {phrase:String}"

### Предопределенный HTTP интерфейс

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

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

RT.WideStore также поддерживает предопределенный HTTP-интерфейс, который может помочь вам легче интегрироваться со сторонними инструментами, такими как [Prometheus exporter](https://github.com/percona-lab/clickhouse_exporter).

Пример:

* Прежде всего, добавьте раздел в конфигурационный файл сервера:

<http\_handlers>

<rule>

<url>/predefined\_query</url>

<methods>POST,GET</methods>

<handler>

<type>predefined\_query\_handler</type>

<query>SELECT \* FROM system.metrics LIMIT 5 FORMAT Template SETTINGS format\_template\_resultset = 'prometheus\_template\_output\_format\_resultset', format\_template\_row = 'prometheus\_template\_output\_format\_row', format\_template\_rows\_between\_delimiter = '\n'</query>

</handler>

</rule>

<rule>...</rule>

<rule>...</rule>

</http\_handlers>

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

$ curl -v 'http://localhost:8123/predefined\_query'

\* Trying ::1...

\* Connected to localhost (::1) port 8123 (#0)

> GET /predefined\_query HTTP/1.1

> Host: localhost:8123

> User-Agent: curl/7.47.0

> Accept: \*/\*

>

< HTTP/1.1 200 OK

< Date: Tue, 28 Apr 2020 08:52:56 GMT

< Connection: Keep-Alive

< Content-Type: text/plain; charset=UTF-8

< X-ClickHouse-Server-Display-Name: i-mloy5trc

< Transfer-Encoding: chunked

< X-ClickHouse-Query-Id: 96fe0052-01e6-43ce-b12a-6b7370de6e8a

< X-ClickHouse-Format: Template

< X-ClickHouse-Timezone: Asia/Shanghai

< Keep-Alive: timeout=3

< X-ClickHouse-Summary: {"read\_rows":"0","read\_bytes":"0","written\_rows":"0","written\_bytes":"0","total\_rows\_to\_read":"0"}

<

# HELP "Query" "Number of executing queries"

# TYPE "Query" counter

"Query" 1

# HELP "Merge" "Number of executing background merges"

# TYPE "Merge" counter

"Merge" 0

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

# TYPE "PartMutation" counter

"PartMutation" 0

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

# TYPE "ReplicatedFetch" counter

"ReplicatedFetch" 0

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

# TYPE "ReplicatedSend" counter

"ReplicatedSend" 0

\* Connection #0 to host localhost left intact

\* Connection #0 to host localhost left intact

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

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

* Method отвечает за соответствие метода HTTP-запроса. Method соответствует методу [method](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods) протокола HTTP. Это необязательная настройка. Если она не определена в файле конфигурации, она не соответствует методу HTTP-запроса.
* url отвечает за соответствие URL HTTP-запроса. Она совместима с регулярными выражениями [RE2](https://github.com/google/re2). Это необязательная настройка. Если она не определена в файле конфигурации, она не соответствует URL-адресу HTTP-запроса.
* headers отвечают за соответствие заголовка HTTP-запроса. Она совместим с регулярными выражениями RE2. Это необязательная настройка. Если она не определен в файле конфигурации, она не соответствует заголовку HTTP-запроса.
* handler содержит основную часть обработчика. Сейчас handler может настраивать type, status, content\_type, response\_content, query, query\_param\_name.

type на данный момент поддерживает три типа: [predefined\_query\_handler](https://clickhouse.tech/docs/ru/interfaces/http/#predefined_query_handler), [dynamic\_query\_handler](https://clickhouse.tech/docs/ru/interfaces/http/#dynamic_query_handler), [static](https://clickhouse.tech/docs/ru/interfaces/http/#static).

* + query – используется с типом predefined\_query\_handler, выполняет запрос при вызове обработчика.
  + query\_param\_name – используется с типом dynamic\_query\_handler, извлекает и выполняет значение, соответствующее значению query\_param\_name в параметрах HTTP-запроса.
  + status – используется с типом static, возвращает код состояния ответа.
  + content\_type – используется с типом static, возвращает [content-type](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Type).
  + response\_content – используется с типомstatic, содержимое ответа, отправленное клиенту, при использовании префикса ‘file://’ or ‘config://’, находит содержимое из файла или конфигурации, отправленного клиенту.

Далее приведены методы настройки для различных типов.

### predefined\_query\_handler

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

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

В следующем примере определяются настройки [max\_threads](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-max_threads) и max\_alter\_threads, а затем запрашивается системная таблица, чтобы проверить, были ли эти параметры успешно установлены.

Пример:

<http\_handlers>

<rule>

<url><![CDATA[/query\_param\_with\_url/\w+/(?P<name\_1>[^/]+)(/(?P<name\_2>[^/]+))?]]></url>

<method>GET</method>

<headers>

<XXX>TEST\_HEADER\_VALUE</XXX>

<PARAMS\_XXX><![CDATA[(?P<name\_1>[^/]+)(/(?P<name\_2>[^/]+))?]]></PARAMS\_XXX>

</headers>

<handler>

<type>predefined\_query\_handler</type>

<query>SELECT value FROM system.settings WHERE name = {name\_1:String}</query>

<query>SELECT name, value FROM system.settings WHERE name = {name\_2:String}</query>

</handler>

</rule>

</http\_handlers>

$ curl -H 'XXX:TEST\_HEADER\_VALUE' -H 'PARAMS\_XXX:max\_threads' 'http://localhost:8123/query\_param\_with\_url/1/max\_threads/max\_alter\_threads?max\_threads=1&max\_alter\_threads=2'

1

max\_alter\_threads 2

Предупреждение

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

### dynamic\_query\_handler

В Dynamic\_query\_handler, запрос пишется в виде параметров HTTP-запроса. Разница в том, что в predefined\_query\_handler, запрос записывается в конфигурационный файл. Вы можете настроить query\_param\_name в dynamic\_query\_handler.

RT.WideStore извлекает и выполняет значение, соответствующее значению query\_param\_name URL-адресе HTTP-запроса. Значение по умолчанию query\_param\_name – это /query. Это необязательная настройка. Если в файле конфигурации нет определения, параметр не передается.

Чтобы поэкспериментировать с этой функциональностью, в примере определяются значения [max\_threads](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-max_threads) и max\_alter\_threads и запрашивается, успешно ли были установлены настройки.

Пример:

<http\_handlers>

<rule>

<headers>

<XXX>TEST\_HEADER\_VALUE\_DYNAMIC</XXX> </headers>

<handler>

<type>dynamic\_query\_handler</type>

<query\_param\_name>query\_param</query\_param\_name>

</handler>

</rule>

</http\_handlers>

$ curl -H 'XXX:TEST\_HEADER\_VALUE\_DYNAMIC' 'http://localhost:8123/own?max\_threads=1&max\_alter\_threads=2&param\_name\_1=max\_threads&param\_name\_2=max\_alter\_threads&query\_param=SELECT%20name,value%20FROM%20system.settings%20where%20name%20=%20%7Bname\_1:String%7D%20OR%20name%20=%20%7Bname\_2:String%7D'

max\_threads 1

max\_alter\_threads 2

### static

Static может возвращать [content\_type](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Type), [status](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status) и response\_content. response\_content может возвращать конкретное содержимое.

Пример:

Возвращает сообщение.

<http\_handlers>

<rule>

<methods>GET</methods>

<headers><XXX>xxx</XXX></headers>

<url>/hi</url>

<handler>

<type>static</type>

<status>402</status>

<content\_type>text/html; charset=UTF-8</content\_type>

<response\_content>Say Hi!</response\_content>

</handler>

</rule>

<http\_handlers>

$ curl -vv -H 'XXX:xxx' 'http://localhost:8123/hi'

\* Trying ::1...

\* Connected to localhost (::1) port 8123 (#0)

> GET /hi HTTP/1.1

> Host: localhost:8123

> User-Agent: curl/7.47.0

> Accept: \*/\*

> XXX:xxx

>

< HTTP/1.1 402 Payment Required

< Date: Wed, 29 Apr 2020 03:51:26 GMT

< Connection: Keep-Alive

< Content-Type: text/html; charset=UTF-8

< Transfer-Encoding: chunked

< Keep-Alive: timeout=3

< X-ClickHouse-Summary: {"read\_rows":"0","read\_bytes":"0","written\_rows":"0","written\_bytes":"0","total\_rows\_to\_read":"0"}

<

\* Connection #0 to host localhost left intact

Say Hi!%

Находит содержимое настроек, отправленных клиенту.

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

<http\_handlers>

<rule>

<methods>GET</methods>

<headers><XXX>xxx</XXX></headers>

<url>/get\_config\_static\_handler</url>

<handler>

<type>static</type>

<response\_content>config://get\_config\_static\_handler</response\_content>

</handler>

</rule>

</http\_handlers>

$ curl -v -H 'XXX:xxx' 'http://localhost:8123/get\_config\_static\_handler'

\* Trying ::1...

\* Connected to localhost (::1) port 8123 (#0)

> GET /get\_config\_static\_handler HTTP/1.1

> Host: localhost:8123

> User-Agent: curl/7.47.0

> Accept: \*/\*

> XXX:xxx

>

< HTTP/1.1 200 OK

< Date: Wed, 29 Apr 2020 04:01:24 GMT

< Connection: Keep-Alive

< Content-Type: text/plain; charset=UTF-8

< Transfer-Encoding: chunked

< Keep-Alive: timeout=3

< X-ClickHouse-Summary: {"read\_rows":"0","read\_bytes":"0","written\_rows":"0","written\_bytes":"0","total\_rows\_to\_read":"0"}

<

\* Connection #0 to host localhost left intact

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

Находит содержимое файла, отправленного клиенту.

<http\_handlers>

<rule>

<methods>GET</methods>

<headers><XXX>xxx</XXX></headers>

<url>/get\_absolute\_path\_static\_handler</url>

<handler>

<type>static</type>

<content\_type>text/html; charset=UTF-8</content\_type>

<response\_content>file:///absolute\_path\_file.html</response\_content>

</handler>

</rule>

<rule>

<methods>GET</methods>

<headers><XXX>xxx</XXX></headers>

<url>/get\_relative\_path\_static\_handler</url>

<handler>

<type>static</type>

<content\_type>text/html; charset=UTF-8</content\_type>

<response\_content>file://./relative\_path\_file.html</response\_content>

</handler>

</rule>

</http\_handlers>

$ user\_files\_path='/var/lib/clickhouse/user\_files'

$ sudo echo "<html><body>Relative Path File</body></html>" > $user\_files\_path/relative\_path\_file.html

$ sudo echo "<html><body>Absolute Path File</body></html>" > $user\_files\_path/absolute\_path\_file.html

$ curl -vv -H 'XXX:xxx' 'http://localhost:8123/get\_absolute\_path\_static\_handler'

\* Trying ::1...

\* Connected to localhost (::1) port 8123 (#0)

> GET /get\_absolute\_path\_static\_handler HTTP/1.1

> Host: localhost:8123

> User-Agent: curl/7.47.0

> Accept: \*/\*

> XXX:xxx

>

< HTTP/1.1 200 OK

< Date: Wed, 29 Apr 2020 04:18:16 GMT

< Connection: Keep-Alive

< Content-Type: text/html; charset=UTF-8

< Transfer-Encoding: chunked

< Keep-Alive: timeout=3

< X-ClickHouse-Summary: {"read\_rows":"0","read\_bytes":"0","written\_rows":"0","written\_bytes":"0","total\_rows\_to\_read":"0"}

<

<html><body>Absolute Path File</body></html>

\* Connection #0 to host localhost left intact

$ curl -vv -H 'XXX:xxx' 'http://localhost:8123/get\_relative\_path\_static\_handler'

\* Trying ::1...

\* Connected to localhost (::1) port 8123 (#0)

> GET /get\_relative\_path\_static\_handler HTTP/1.1

> Host: localhost:8123

> User-Agent: curl/7.47.0

> Accept: \*/\*

> XXX:xxx

>

< HTTP/1.1 200 OK

< Date: Wed, 29 Apr 2020 04:18:31 GMT

< Connection: Keep-Alive

< Content-Type: text/html; charset=UTF-8

< Transfer-Encoding: chunked

< Keep-Alive: timeout=3

< X-ClickHouse-Summary: {"read\_rows":"0","read\_bytes":"0","written\_rows":"0","written\_bytes":"0","total\_rows\_to\_read":"0"}

<

<html><body>Relative Path File</body></html>

\* Connection #0 to host localhost left intact

### MySQL-интерфейс

RT.WideStore поддерживает взаимодействие по протоколу MySQL. Данная функция включается настройкой [mysql\_port](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/#server_configuration_parameters-mysql_port) в конфигурационном файле:

<mysql\_port>9004</mysql\_port>

Пример подключения с помощью стандартного клиента mysql:

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

Вывод в случае успешного подключения:

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 4

Server version: 20.2.1.1-ClickHouse

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its

affiliates. Other names may be trademarks of their respective

owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>

Для совместимости со всеми клиентами рекомендуется задавать пароль пользователя в конфигурационном файле с помощью двойного хэша [SHA1](https://clickhouse.tech/docs/ru/operations/settings/settings-users/#password_double_sha1_hex).

В случае указания пароля с помощью [SHA256](https://clickhouse.tech/docs/ru/operations/settings/settings-users/#password_sha256_hex) некоторые клиенты не смогут пройти аутентификацию (mysqljs и старые версии стандартного клиента mysql).

Ограничения:

* не поддерживаются подготовленные запросы,
* некоторые типы данных отправляются как строки.

## Движки таблиц

Движок таблицы (тип таблицы) определяет:

* как и где хранятся данные, куда их писать и откуда читать,
* какие запросы поддерживаются и каким образом,
* конкурентный доступ к данным,
* использование индексов, если есть,
* возможно ли многопоточное выполнение запроса,
* параметры репликации данных.

### Семейства движков

### MergeTree

Наиболее универсальные и функциональные движки таблиц для задач с высокой загрузкой. Общим свойством этих движков является быстрая вставка данных с последующей фоновой обработкой данных. Движки \*MergeTree поддерживают репликацию данных (в [Replicated\*](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/replication/#replication) версиях движков), партиционирование, и другие возможности не поддержанные для других движков.

Движки семейства:

* [MergeTree](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/#mergetree),
* [ReplacingMergeTree](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/replacingmergetree/#replacingmergetree),
* [SummingMergeTree](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/summingmergetree/#summingmergetree),
* [AggregatingMergeTree](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/aggregatingmergetree/#aggregatingmergetree),
* [CollapsingMergeTree](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/collapsingmergetree/#table_engine-collapsingmergetree),
* [VersionedCollapsingMergeTree](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/versionedcollapsingmergetree/#versionedcollapsingmergetree),
* [GraphiteMergeTree](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/graphitemergetree/#graphitemergetree).

### Log

Простые [движки](https://clickhouse.tech/docs/ru/engines/table-engines/log-family/) с минимальной функциональностью. Они наиболее эффективны, когда вам нужно быстро записать много небольших таблиц (до примерно 1 миллиона строк) и прочитать их позже целиком.

Движки семейства:

* [TinyLog](https://clickhouse.tech/docs/ru/engines/table-engines/log-family/tinylog/#tinylog),
* [StripeLog](https://clickhouse.tech/docs/ru/engines/table-engines/log-family/stripelog/#stripelog),
* [Log](https://clickhouse.tech/docs/ru/engines/table-engines/log-family/log/#log).

### Движки для интеграции

Движки для связи с другими системами хранения и обработки данных.

Движки семейства:

* [Kafka](https://clickhouse.tech/docs/ru/engines/table-engines/integrations/kafka/#kafka),
* [MySQL](https://clickhouse.tech/docs/ru/engines/table-engines/integrations/mysql/#mysql),
* [ODBC](https://clickhouse.tech/docs/ru/engines/table-engines/integrations/odbc/#table-engine-odbc),
* [JDBC](https://clickhouse.tech/docs/ru/engines/table-engines/integrations/jdbc/#table-engine-jdbc).

### Специальные движки

Движки семейства:

* [Distributed](https://clickhouse.tech/docs/ru/engines/table-engines/special/distributed/#distributed),
* [MaterializedView](https://clickhouse.tech/docs/ru/engines/table-engines/special/materializedview/#materializedview),
* [Dictionary](https://clickhouse.tech/docs/ru/engines/table-engines/special/dictionary/#dictionary),
* [Merge](https://clickhouse.tech/docs/ru/engines/table-engines/special/merge/#merge),
* [File](https://clickhouse.tech/docs/ru/engines/table-engines/special/file/#file),
* [Null](https://clickhouse.tech/docs/ru/engines/table-engines/special/null/#null),
* [Set](https://clickhouse.tech/docs/ru/engines/table-engines/special/set/#set),
* [Join](https://clickhouse.tech/docs/ru/engines/table-engines/special/join/#join),
* [URL](https://clickhouse.tech/docs/ru/engines/table-engines/special/url/#table_engines-url),
* [View](https://clickhouse.tech/docs/ru/engines/table-engines/special/view/#table_engines-view),
* [Memory](https://clickhouse.tech/docs/ru/engines/table-engines/special/memory/#memory),
* [Buffer](https://clickhouse.tech/docs/ru/engines/table-engines/special/buffer/#buffer).

### Виртуальные столбцы

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

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

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

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

## Справка по SQL в RT.WideStore

### SQL выражения в RT.WideStore

Выражения описывают различные действия, которые можно выполнить с помощью SQL запросов. Каждый вид выражения имеет свой синтаксис и особенности использования, которые описаны в *Приложении 1*:

* [SELECT](https://clickhouse.tech/docs/ru/sql-reference/statements/select/),
* [INSERT INTO](https://clickhouse.tech/docs/ru/sql-reference/statements/insert-into/),
* [CREATE](https://clickhouse.tech/docs/ru/sql-reference/statements/create/),
* [ALTER](https://clickhouse.tech/docs/ru/sql-reference/statements/alter/),
* [SYSTEM](https://clickhouse.tech/docs/ru/sql-reference/statements/system/),
* [SHOW](https://clickhouse.tech/docs/ru/sql-reference/statements/show/),
* [GRANT](https://clickhouse.tech/docs/ru/sql-reference/statements/grant/),
* [REVOKE](https://clickhouse.tech/docs/ru/sql-reference/statements/revoke/),
* [ATTACH](https://clickhouse.tech/docs/ru/sql-reference/statements/attach/),
* [CHECK TABLE](https://clickhouse.tech/docs/ru/sql-reference/statements/check-table/),
* [DESCRIBE TABLE](https://clickhouse.tech/docs/ru/sql-reference/statements/describe-table/),
* [DETACH](https://clickhouse.tech/docs/ru/sql-reference/statements/detach/),
* [DROP](https://clickhouse.tech/docs/ru/sql-reference/statements/drop/),
* [EXISTS](https://clickhouse.tech/docs/ru/sql-reference/statements/exists/),
* [KILL](https://clickhouse.tech/docs/ru/sql-reference/statements/kill/),
* [OPTIMIZE](https://clickhouse.tech/docs/ru/sql-reference/statements/optimize/),
* [RENAME](https://clickhouse.tech/docs/ru/sql-reference/statements/rename/),
* [SET](https://clickhouse.tech/docs/ru/sql-reference/statements/set/),
* [SET ROLE](https://clickhouse.tech/docs/ru/sql-reference/statements/set-role/),
* [TRUNCATE](https://clickhouse.tech/docs/ru/sql-reference/statements/truncate/),
* [USE](https://clickhouse.tech/docs/ru/sql-reference/statements/use/).

### Синтаксис

В системе есть два вида парсеров: полноценный парсер SQL (recursive descent parser) и парсер форматов данных (быстрый потоковый парсер).

Во всех случаях кроме запроса INSERT, используется только полноценный парсер SQL.

В запросе INSERT используется оба парсера:

**INSERT** **INTO** t **VALUES** (1, 'Hello, world'), (2, 'abc'), (3, 'def')

Фрагмент INSERT INTO t VALUES парсится полноценным парсером, а данные (1, 'Hello, world'), (2, 'abc'), (3, 'def') - быстрым потоковым парсером.

Данные могут иметь любой формат. При получении запроса, сервер заранее считывает в оперативку не более max\_query\_size байт запроса (по умолчанию, 1МБ), а всё остальное обрабатывается потоково.

Таким образом, в системе нет проблем с большими INSERT запросами, как в MySQL.

При использовании формата Values в INSERT запросе может сложиться иллюзия, что данные парсятся также, как выражения в запросе SELECT, но это не так. Формат Values гораздо более ограничен.

Далее пойдёт речь о полноценном парсере. О парсерах форматов, смотри раздел «Форматы».

### Пробелы

Между синтаксическими конструкциями (в том числе, в начале и конце запроса) может быть расположено произвольное количество пробельных символов. К пробельным символам относятся пробел, таб, перевод строки, CR, form feed.

### Комментарии

Поддерживаются комментарии в SQL-стиле и C-стиле.

Комментарии в SQL-стиле: от -- до конца строки. Пробел после -- может не ставиться.

Комментарии в C-стиле: от /\* до \*/. Такие комментарии могут быть многострочными. Пробелы тоже не обязательны.

### Ключевые слова

Ключевые слова не зависят от регистра, если они соответствуют:

* Стандарту SQL. Например, применение любого из вариантов SELECT, select или SeLeCt не вызовет ошибки.
* Реализации в некоторых популярных DBMS (MySQL или Postgres). Например, DateTime и datetime.

Зависимость от регистра для имён типов данных можно проверить в таблице [system.data\_type\_families](https://clickhouse.tech/docs/ru/operations/system-tables/data_type_families/#system_tables-data_type_families).

В отличие от стандарта SQL, все остальные ключевые слова, включая названия функций зависят от регистра.

Ключевые слова не зарезервированы (а всего лишь парсятся как ключевые слова в соответствующем контексте). Если вы используете [идентификаторы](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-identifiers), совпадающие с ключевыми словами, заключите их в кавычки. Например, запрос SELECT "FROM" FROM table\_name валиден, если таблица table\_name имеет столбец с именем "FROM".

### Идентификаторы

Идентификаторы:

* имена кластеров, баз данных, таблиц, разделов и столбцов,
* функции,
* типы данных,
* [синонимы выражений](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-expression_aliases).

Некоторые идентификаторы нужно указывать в кавычках (например, идентификаторы с пробелами). Прочие идентификаторы можно указывать без кавычек. Рекомендуется использовать идентификаторы, не требующие кавычек.

Идентификаторы не требующие кавычек соответствуют регулярному выражению ^[a-zA-Z\_][0-9a-zA-Z\_]\*$ и не могут совпадать с [ключевыми словами](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-keywords). Примеры: x, \_1, X\_y\_\_Z123\_.

Если вы хотите использовать идентификаторы, совпадающие с ключевыми словами, или использовать в идентификаторах символы, не входящие в регулярное выражение, заключите их в двойные или обратные кавычки, например, "id", `id`.

### Литералы

Существуют: числовые, строковые, составные литералы и NULL.

### Числовые

Числовой литерал пытается распарситься:

1. Сначала как знаковое 64-разрядное число, функцией [strtoull](https://en.cppreference.com/w/cpp/string/byte/strtoul).
2. Если не получилось, то как беззнаковое 64-разрядное число, функцией [strtoll](https://en.cppreference.com/w/cpp/string/byte/strtol).
3. Если не получилось, то как число с плавающей запятой, функцией [strtod](https://en.cppreference.com/w/cpp/string/byte/strtof).
4. Иначе – ошибка.

Соответствующее значение будет иметь тип минимального размера, который вмещает значение.

Например, 1 парсится как UInt8, а 256 как UInt16. Подробнее о типах данных читайте в разделе [Типы данных](https://clickhouse.tech/docs/ru/sql-reference/syntax/).

Примеры: 1, 18446744073709551615, 0xDEADBEEF, 01, 0.1, 1e100, -1e-100, inf, nan.

### Строковые

Поддерживаются только строковые литералы в одинарных кавычках. Символы внутри могут быть экранированы с помощью обратного слеша. Следующие escape-последовательности имеют соответствующее специальное значение: \b, \f, \r, \n, \t, \0, \a, \v, \xHH. Во всех остальных случаях, последовательности вида \c, где c – любой символ, преобразуется в c. Таким образом, могут быть использованы последовательности \' и \\. Значение будет иметь тип [String](https://clickhouse.tech/docs/ru/sql-reference/syntax/).

Минимальный набор символов, которых вам необходимо экранировать в строковых литералах: ' и \. Одинарная кавычка может быть экранирована одинарной кавычкой, литералы 'It\'s' и 'It''s' эквивалентны.

### Составные

Поддерживаются конструкции для массивов: [1, 2, 3] и кортежей: (1, 'Hello, world!', 2).

На самом деле, это вовсе не литералы, а выражение с оператором создания массива и оператором создания кортежа, соответственно.

Массив должен состоять хотя бы из одного элемента, а кортеж - хотя бы из двух.

Кортежи носят служебное значение для использования в секции IN запроса SELECT. Кортежи могут быть получены как результат запроса, но они не могут быть сохранены в базе данных (за исключением таблицы [Memory](https://clickhouse.tech/docs/ru/sql-reference/syntax/).)

### NULL

Обозначает, что значение отсутствует.

Чтобы в поле таблицы можно было хранить NULL, оно должно быть типа [Nullable](https://clickhouse.tech/docs/ru/sql-reference/syntax/).

В зависимости от формата данных (входных или выходных) NULL может иметь различное представление. Подробнее смотрите в документации для [форматов данных](https://clickhouse.tech/docs/ru/interfaces/formats/#formats).

При обработке NULL есть множество особенностей. Например, если хотя бы один из аргументов операции сравнения – NULL, то результатом такой операции тоже будет NULL. Этим же свойством обладают операции умножения, сложения и пр. Подробнее читайте в документации на каждую операцию.

В запросах можно проверить NULL с помощью операторов [IS NULL](https://clickhouse.tech/docs/ru/sql-reference/operators/#operator-is-null) и [IS NOT NULL](https://clickhouse.tech/docs/ru/sql-reference/operators/), а также соответствующих функций isNull и isNotNull.

### Функции, синтаксис

Функции записываются как идентификатор со списком аргументов (возможно, пустым) в скобках. В отличие от стандартного SQL, даже в случае пустого списка аргументов, скобки обязательны. Пример: now().

Бывают обычные и агрегатные функции (смотрите раздел «Агрегатные функции»). Некоторые агрегатные функции могут содержать два списка аргументов в круглых скобках. Пример: quantile(0.9)(x).

Такие агрегатные функции называются «параметрическими», а первый список аргументов называется «параметрами». Синтаксис агрегатных функций без параметров ничем не отличается от обычных функций.

Описание функций представлено в *Приложении 2*.

### Операторы

Операторы преобразуются в соответствующие им функции во время парсинга запроса, с учётом их приоритета и ассоциативности.

Например, выражение 1 + 2 \* 3 + 4 преобразуется в plus(plus(1, multiply(2, 3)), 4).

### Типы данных и движки таблиц

Типы данных и движки таблиц в запросе CREATE записываются также, как идентификаторы или также как функции. То есть, могут содержать или не содержать список аргументов в круглых скобках. Подробнее смотрите разделы «Типы данных», «Движки таблиц», «CREATE».

### Синонимы выражений

Синоним – это пользовательское имя выражения в запросе:

expr **AS** **alias**

Где:

* AS – ключевое слово для определения синонимов. Можно определить синоним для имени таблицы или столбца в секции SELECT без использования ключевого слова AS.

Например, `SELECT table\_name\_alias.column\_name FROM table\_name table\_name\_alias`.

В функции [CAST](sql\_reference/syntax.md#type\_conversion\_function-cast), ключевое слово `AS` имеет другое значение. Смотрите описание функции.

* expr – любое выражение, которое поддерживает RT.WideStore.

Например, `SELECT column\_name \* 2 AS double FROM some\_table`.

* alias – имя для выражения. Синонимы должны соответствовать синтаксису [идентификаторов](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-identifiers).

Например, `SELECT "table t".column\_name FROM table\_name AS "table t"`.

### Примечания по использованию

Синонимы являются глобальными для запроса или подзапроса, и вы можете определить синоним в любой части запроса для любого выражения. Например, SELECT (1 AS n) + 2, n.

Синонимы не передаются в подзапросы и между подзапросами. Например, при выполнении запроса SELECT (SELECT sum(b.a) + num FROM b) - a.a AS num FROM a RT.WideStore сгенерирует исключение Unknown identifier: num.

Если синоним определен для результирующих столбцов в секции SELECT вложенного запроса, то эти столбцы отображаются во внешнем запросе. Например, SELECT n + m FROM (SELECT 1 AS n, 2 AS m).

Будьте осторожны с синонимами, совпадающими с именами столбцов или таблиц. Рассмотрим следующий пример:

**CREATE** **TABLE** t

(

a Int,

b Int

)

ENGINE = TinyLog()

**SELECT**

argMax(a, b),

**sum**(b) **AS** b

**FROM** t

Received exception from server (version 18.14.17):

Code: 184. DB::Exception: Received from localhost:9000, 127.0.0.1. DB::Exception: Aggregate function sum(b) is found inside another aggregate function in query.

В этом примере мы объявили таблицу t со столбцом b. Затем, при выборе данных, мы определили синоним sum(b) AS b. Поскольку синонимы глобальные, то RT.WideStore заменил литерал b в выражении argMax(a, b) выражением sum(b). Эта замена вызвала исключение.

### Звёздочка

В запросе SELECT, вместо выражения может стоять звёздочка. Подробнее смотрите раздел «SELECT».

### Выражения

Выражение представляет собой функцию, идентификатор, литерал, применение оператора, выражение в скобках, подзапрос, звёздочку. А также может содержать синоним.

Список выражений – одно выражение или несколько выражений через запятую.

Функции и операторы, в свою очередь, в качестве аргументов, могут иметь произвольные выражения.

### Распределенные DDL запросы (секция ON CLUSTER)

Запросы CREATE, DROP, ALTER, RENAME поддерживают возможность распределенного выполнения на кластере.

Например, следующий запрос создает распределенную (Distributed) таблицу all\_hits на каждом хосте в cluster:

**CREATE** **TABLE** **IF** **NOT** **EXISTS** all\_hits **ON** **CLUSTER** **cluster** (p Date, i Int32) ENGINE = Distributed(**cluster**, **default**, hits)

Для корректного выполнения таких запросов необходимо на каждом хосте иметь одинаковое определение кластера (для упрощения синхронизации конфигов можете использовать подстановки из ZooKeeper). Также необходимо подключение к ZooKeeper серверам.

Локальная версия запроса в конечном итоге будет выполнена на каждом хосте кластера, даже если некоторые хосты в данный момент не доступны. Гарантируется упорядоченность выполнения запросов в рамках одного хоста.

### Функции

Функции бывают как минимум двух видов – обычные функции (называются просто, функциями) и агрегатные функции. Это совершенно разные вещи. Обычные функции работают так, как будто применяются к каждой строке по отдельности (для каждой строки, результат вычисления функции не зависит от других строк). Агрегатные функции аккумулируют множество значений из разных строк (то есть, зависят от целого множества строк).

В этом разделе речь пойдёт об обычных функциях. Для агрегатных функций, смотрите раздел «Агрегатные функции».

***Примечание:*** *Есть ещё третий вид функций, к которым относится функция arrayJoin; также можно отдельно иметь ввиду табличные функции.*

### Строгая типизация

В RT.WideStore, в отличие от стандартного SQL, типизация является строгой. То есть, не производится неявных преобразований между типами. Все функции работают для определённого набора типов. Это значит, что иногда вам придётся использовать функции преобразования типов.

### Склейка одинаковых выражений

Все выражения в запросе, имеющие одинаковые AST (одинаковую запись или одинаковый результат синтаксического разбора), считаются имеющими одинаковые значения. Такие выражения склеиваются и исполняются один раз. Одинаковые подзапросы тоже склеиваются.

### Типы результата

Все функции возвращают одно (не несколько, не ноль) значение в качестве результата. Тип результата обычно определяется только типами аргументов, но не значениями аргументов. Исключение - функция tupleElement (оператор a.N), а также функция toFixedString.

### Константы

Для простоты, некоторые функции могут работать только с константами в качестве некоторых аргументов. Например, правый аргумент оператора LIKE должен быть константой.

Почти все функции возвращают константу для константных аргументов. Исключение – функции генерации случайных чисел.

Функция now возвращает разные значения для запросов, выполненных в разное время, но результат считается константой, так как константность важна лишь в пределах одного запроса.

Константное выражение также считается константой (например, правую часть оператора LIKE можно сконструировать из нескольких констант).

Функции могут быть по-разному реализованы для константных и не константных аргументов (выполняется разный код). Но результат работы для константы и полноценного столбца, содержащего только одно такое же значение, должен совпадать.

Обработка NULL

Функции имеют следующие виды поведения:

* Если хотя бы один из аргументов функции – NULL, то результат функции тоже NULL.
* Специальное поведение, указанное в описании каждой функции отдельно. В исходном коде RT.WideStore такие функции можно определить по свойству UseDefaultImplementationForNulls=false.

### Неизменяемость

Функции не могут поменять значения своих аргументов - любые изменения возвращаются в качестве результата. Соответственно, от порядка записи функций в запросе, результат вычислений отдельных функций не зависит.

### Функции высшего порядка, оператор -> и функция lambda(params, expr)

Функции высшего порядка, в качестве своего функционального аргумента могут принимать только лямбда-функции. Чтобы передать лямбда-функцию в функцию высшего порядка, используйте оператор ->. Слева от стрелочки стоит формальный параметр – произвольный идентификатор, или несколько формальных параметров – произвольные идентификаторы в кортеже. Справа от стрелочки стоит выражение, в котором могут использоваться эти формальные параметры, а также любые столбцы таблицы.

Примеры:

x -> 2 \* x

str -> str != Referer

В функции высшего порядка может быть передана лямбда-функция, принимающая несколько аргументов. В этом случае в функцию высшего порядка передаётся несколько массивов одинаковой длины, которым эти аргументы будут соответствовать.

Для некоторых функций первый аргумент (лямбда-функция) может отсутствовать. В этом случае подразумевается тождественное отображение.

### Обработка ошибок

Некоторые функции могут кидать исключения в случае ошибочных данных. В этом случае, выполнение запроса прерывается, и текст ошибки выводится клиенту. При распределённой обработке запроса, при возникновении исключения на одном из серверов, на другие серверы пытается отправиться просьба тоже прервать выполнение запроса.

### Вычисление выражений-аргументов

В почти всех языках программирования, для некоторых операторов может не вычисляться один из аргументов. Обычно – для операторов &&, ||, ?:.

Но в RT.WideStore, аргументы функций (операторов) вычисляются всегда. Это связано с тем, что вычисления производятся не по отдельности для каждой строки, а сразу для целых кусочков столбцов.

### Выполнение функций при распределённой обработке запроса

При распределённой обработке запроса, как можно большая часть стадий выполнения запроса производится на удалённых серверах, а оставшиеся стадии (слияние промежуточных результатов и всё, что дальше) - на сервере-инициаторе запроса.

Это значит, что выполнение функций может производиться на разных серверах.

Например, в запросе SELECT f(sum(g(x))) FROM distributed\_table GROUP BY h(y),

* если distributed\_table имеет хотя бы два шарда, то функции g и h выполняются на удалённых серверах, а функция f - на сервере-инициаторе запроса;
* если distributed\_table имеет только один шард, то все функции f, g, h выполняются на сервере этого шарда.

Обычно результат выполнения функции не зависит от того, на каком сервере её выполнить. Но иногда это довольно важно.

Например, функции, работающие со словарями, будут использовать словарь, присутствующий на том сервере, на котором они выполняются.

Другой пример – функция hostName вернёт имя сервера, на котором она выполняется, и это можно использовать для служебных целей - чтобы в запросе SELECT сделать GROUP BY по серверам.

Если функция в запросе выполняется на сервере-инициаторе запроса, а вам нужно, чтобы она выполнялась на удалённых серверах, вы можете обернуть её в агрегатную функцию any или добавить в ключ в GROUP BY.

ПРИЛОЖЕНИЕ 1. ОПИСАНИЕ SQL ВЫРАЖЕНИЙ В RT.WIDESTORE

##### Синтаксис запросов SELECT

SELECT выполняет получение данных.

[**WITH** expr\_list|(subquery)]

**SELECT** [**DISTINCT**] expr\_list

[**FROM** [db.]**table** | (subquery) | table\_function] [**FINAL**]

[SAMPLE sample\_coeff]

[ARRAY **JOIN** ...]

[**GLOBAL**] [**ANY**|**ALL**|ASOF] [**INNER**|**LEFT**|**RIGHT**|**FULL**|**CROSS**] [**OUTER**|SEMI|ANTI] **JOIN** (subquery)|**table** (**ON** <expr\_list>)|(**USING** <column\_list>)

[PREWHERE expr]

[**WHERE** expr]

[**GROUP** **BY** expr\_list] [**WITH** **ROLLUP**|**WITH** **CUBE**] [**WITH** TOTALS]

[**HAVING** expr]

[**ORDER** **BY** expr\_list] [**WITH** FILL] [**FROM** expr] [**TO** expr] [STEP expr]

[**LIMIT** [offset\_value, ]n **BY** columns]

[**LIMIT** [n, ]m] [**WITH** TIES]

[SETTINGS ...]

[**UNION** **ALL** ...]

[**INTO** OUTFILE filename]

[FORMAT format]

Все секции являются необязательными, за исключением списка выражений сразу после SELECT, о котором более подробно будет рассказано [ниже](https://clickhouse.tech/docs/ru/sql-reference/statements/select/#select-clause).

Особенности каждой необязательной секции рассматриваются в отдельных разделах, которые перечислены в том же порядке, в каком они выполняются:

* [секция WITH](https://clickhouse.tech/docs/ru/sql-reference/statements/select/with/);
* [секция FROM](https://clickhouse.tech/docs/ru/sql-reference/statements/select/from/);
* [секция SAMPLE](https://clickhouse.tech/docs/ru/sql-reference/statements/select/sample/);
* [секция JOIN](https://clickhouse.tech/docs/ru/sql-reference/statements/select/join/);
* [секция PREWHERE](https://clickhouse.tech/docs/ru/sql-reference/statements/select/prewhere/);
* [секция WHERE](https://clickhouse.tech/docs/ru/sql-reference/statements/select/where/);
* [секция GROUP BY](https://clickhouse.tech/docs/ru/sql-reference/statements/select/group-by/);
* [секция LIMIT BY](https://clickhouse.tech/docs/ru/sql-reference/statements/select/limit-by/);
* [секция HAVING](https://clickhouse.tech/docs/ru/sql-reference/statements/select/having/);
* [секция SELECT](https://clickhouse.tech/docs/ru/sql-reference/statements/select/#select-clause);
* [секция DISTINCT](https://clickhouse.tech/docs/ru/sql-reference/statements/select/distinct/);
* [секция LIMIT](https://clickhouse.tech/docs/ru/sql-reference/statements/select/limit/);
* [секция UNION ALL](https://clickhouse.tech/docs/ru/sql-reference/statements/select/union/);
* [секция INTO OUTFILE](https://clickhouse.tech/docs/ru/sql-reference/statements/select/into-outfile/);
* [секция FORMAT](https://clickhouse.tech/docs/ru/sql-reference/statements/select/format/).

###### Секция SELECT

[Выражения](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-expressions) указанные в секции SELECT анализируются после завершения всех вычислений из секций, описанных выше. Вернее, анализируются выражения, стоящие над агрегатными функциями, если есть агрегатные функции.

Сами агрегатные функции и то, что под ними, вычисляются при агрегации (GROUP BY). Эти выражения работают так, как будто применяются к отдельным строкам результата.

Если в результат необходимо включить все столбцы, используйте символ звёздочка (\*). Например, SELECT \* FROM ....

Чтобы включить в результат несколько столбцов, выбрав их имена с помощью регулярных выражений [re2](https://en.wikipedia.org/wiki/RE2_(software)), используйте выражение COLUMNS.

COLUMNS('regexp')

Например, рассмотрим таблицу:

**CREATE** **TABLE** **default**.col\_names (aa Int8, ab Int8, bc Int8) ENGINE = TinyLog

Следующий запрос выбирает данные из всех столбцов, содержащих в имени символ a.

**SELECT** COLUMNS('a') **FROM** col\_names

┌─aa─┬─ab─┐

│ 1 │ 1 │

└────┴────┘

Выбранные стоблцы возвращаются не в алфавитном порядке.

В запросе можно использовать несколько выражений COLUMNS, а также вызывать над ними функции.

Например:

**SELECT** COLUMNS('a'), COLUMNS('c'), toTypeName(COLUMNS('c')) **FROM** col\_names

┌─aa─┬─ab─┬─bc─┬─toTypeName(bc)─┐

│ 1 │ 1 │ 1 │ Int8 │

└────┴────┴────┴────────────────┘

Каждый столбец, возвращённый выражением COLUMNS, передаётся в функцию отдельным аргументом. Также можно передавать и другие аргументы, если функция их поддерживаем. Аккуратно используйте функции. Если функция не поддерживает переданное количество аргументов, то RT.WideStore генерирует исключение.

Например:

**SELECT** COLUMNS('a') + COLUMNS('c') **FROM** col\_names

Received exception from server (version 19.14.1):

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

В этом примере, COLUMNS('a') возвращает два столбца: aa и ab. COLUMNS('c') возвращает столбец bc. Оператор + не работает с тремя аргументами, поэтому RT.WideStore генерирует исключение с соответствущим сообщением.

Столбцы, которые возвращаются выражением COLUMNS могут быть разных типов. Если COLUMNS не возвращает ни одного столбца и это единственное выражение в запросе SELECT, то RT.WideStore генерирует исключение.

###### Звёздочка

В любом месте запроса, вместо выражения, может стоять звёздочка. При анализе запроса звёздочка раскрывается в список всех столбцов таблицы (за исключением MATERIALIZED и ALIAS столбцов). Есть лишь немного случаев, когда оправдано использовать звёздочку:

* при создании дампа таблицы;
* для таблиц, содержащих всего несколько столбцов - например, системных таблиц;
* для получения информации о том, какие столбцы есть в таблице; в этом случае, укажите LIMIT 1. Но лучше используйте запрос DESC TABLE;
* при наличии сильной фильтрации по небольшому количеству столбцов с помощью PREWHERE;
* в подзапросах (так как из подзапросов выкидываются столбцы, не нужные для внешнего запроса).

В других случаях использование звёздочки является издевательством над системой, так как вместо преимуществ столбцовой СУБД вы получаете недостатки. То есть использовать звёздочку не рекомендуется.

###### Экстремальные значения

Вы можете получить в дополнение к результату также минимальные и максимальные значения по столбцам результата. Для этого выставите настройку **extremes** в 1. Минимумы и максимумы считаются для числовых типов, дат, дат-с-временем. Для остальных столбцов будут выведены значения по умолчанию.

Вычисляются дополнительные две строчки – минимумы и максимумы, соответственно. Эти две дополнительные строки выводятся в [форматах](https://clickhouse.tech/docs/ru/interfaces/formats/) JSON\*, TabSeparated\*, и Pretty\* отдельно от остальных строчек. В остальных форматах они не выводится.

Во форматах JSON\*, экстремальные значения выводятся отдельным полем ‘extremes’. В форматах TabSeparated\*, строка выводится после основного результата и после ‘totals’ если есть. Перед ней (после остальных данных) вставляется пустая строка. В форматах Pretty\*, строка выводится отдельной таблицей после основного результата и после totals если есть.

Экстремальные значения вычисляются для строк перед LIMIT, но после LIMIT BY. Однако при использовании LIMIT offset, size, строки перед offset включаются в extremes. В потоковых запросах, в результате может учитываться также небольшое количество строчек, прошедших LIMIT.

###### Замечания

Вы можете использовать синонимы (алиасы AS) в любом месте запроса.

В секциях GROUP BY, ORDER BY, в отличие от диалекта MySQL, и в соответствии со стандартным SQL, не поддерживаются позиционные аргументы.

Например, если вы напишите GROUP BY 1, 2 - то это будет воспринято, как группировка по константам (то есть, агрегация всех строк в одну).

###### Детали реализации

Если в запросе отсутствуют секции DISTINCT, GROUP BY, ORDER BY, подзапросы в IN и JOIN, то запрос будет обработан полностью потоково, с использованием O(1) количества оперативки.

Иначе запрос может съесть много оперативки, если не указаны подходящие ограничения:

* max\_memory\_usage,
* max\_rows\_to\_group\_by,
* max\_rows\_to\_sort,
* max\_rows\_in\_distinct,
* max\_bytes\_in\_distinct,
* max\_rows\_in\_set,
* max\_bytes\_in\_set,
* max\_rows\_in\_join,
* max\_bytes\_in\_join,
* max\_bytes\_before\_external\_sort,
* max\_bytes\_before\_external\_group\_by.

Подробнее смотрите в разделе «3.10 Настройки» документа «RT.WideStore. Руководство администратора». Присутствует возможность использовать внешнюю сортировку (с сохранением временных данных на диск) и внешнюю агрегацию.

###### Модификаторы запроса SELECT

Вы можете использовать следующие модификаторы в запросах SELECT.

APPLY

Вызывает указанную функцию для каждой строки, возвращаемой внешним табличным выражением запроса.

Синтаксис:

**SELECT** <expr> APPLY( <func> ) **FROM** [db.]**table\_name**

Пример:

**CREATE** **TABLE** columns\_transformers (i Int64, j Int16, k Int64) ENGINE = MergeTree **ORDER** **by** (i);

**INSERT** **INTO** columns\_transformers **VALUES** (100, 10, 324), (120, 8, 23);

**SELECT** \* APPLY(**sum**) **FROM** columns\_transformers;

┌─sum(i)─┬─sum(j)─┬─sum(k)─┐

│ 220 │ 18 │ 347 │

└────────┴────────┴────────┘

EXCEPT

Исключает из результата запроса один или несколько столбцов.

Синтаксис:

**SELECT** <expr> **EXCEPT** ( col\_name1 [, col\_name2, col\_name3, ...] ) **FROM** [db.]**table\_name**

Пример:

**SELECT** \* **EXCEPT** (i) **from** columns\_transformers;

┌──j─┬───k─┐

│ 10 │ 324 │

│ 8 │ 23 │

└────┴─────┘

REPLACE

Определяет одно или несколько [выражений алиасов](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-expression_aliases). Каждый алиас должен соответствовать имени столбца из запроса SELECT \*. В списке столбцов результата запроса имя столбца, соответствующее алиасу, заменяется выражением в модификаторе REPLACE.

Этот модификатор не изменяет имена или порядок столбцов. Однако он может изменить значение и тип значения.

Синтаксис:

**SELECT** <expr> **REPLACE**( <expr> **AS** col\_name) **from** [db.]**table\_name**

Пример:

**SELECT** \* **REPLACE**(i + 1 **AS** i) **from** columns\_transformers;

┌───i─┬──j─┬───k─┐

│ 101 │ 10 │ 324 │

│ 121 │ 8 │ 23 │

└─────┴────┴─────┘

Комбинации модификаторов:

Вы можете использовать каждый модификатор отдельно или комбинировать их.

Примеры:

Использование одного и того же модификатора несколько раз.

**SELECT** COLUMNS('[jk]') APPLY(toString) APPLY(**length**) APPLY(**max**) **from** columns\_transformers;

┌─max(length(toString(j)))─┬─max(length(toString(k)))─┐

│ 2 │ 3 │

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

Использование нескольких модификаторов в одном запросе.

**SELECT** \* **REPLACE**(i + 1 **AS** i) **EXCEPT** (j) APPLY(**sum**) **from** columns\_transformers;

┌─sum(plus(i, 1))─┬─sum(k)─┐

│ 222 │ 347 │

└─────────────────┴────────┘

###### SETTINGS в запросе SELECT

Вы можете задать значения необходимых настроек непосредственно в запросе SELECT в секции SETTINGS. Эти настройки действуют только в рамках данного запроса, а после его выполнения сбрасываются до предыдущего значения или значения по умолчанию.

Другие способы задания настроек описаны [здесь](https://clickhouse.tech/docs/ru/operations/settings/).

Пример:

**SELECT** \* **FROM** some\_table SETTINGS optimize\_read\_in\_order=1, cast\_keep\_nullable=1;

##### INSERT

Добавление данных.

Базовый формат запроса:

**INSERT** **INTO** [db.]**table** [(c1, c2, c3)] **VALUES** (v11, v12, v13), (v21, v22, v23), ...

Вы можете указать список столбцов для вставки, используя синтаксис (c1, c2, c3). Также можно использовать выражение cо [звездочкой](https://clickhouse.tech/docs/ru/sql-reference/statements/select/#asterisk) и/или модификаторами, такими как [APPLY](https://clickhouse.tech/docs/ru/sql-reference/statements/select/#apply-modifier), [EXCEPT](https://clickhouse.tech/docs/ru/sql-reference/statements/select/#except-modifier), [REPLACE](https://clickhouse.tech/docs/ru/sql-reference/statements/select/#replace-modifier).

В качестве примера рассмотрим таблицу:

**SHOW** **CREATE** insert\_select\_testtable

┌─statement────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────┐

│ **CREATE** **TABLE** insert\_select\_testtable

(

`a` **Int8**,

`b` String,

`c` **Int8**

)

ENGINE = MergeTree()

**ORDER** **BY** a │

└──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────┘

**INSERT** **INTO** insert\_select\_testtable (\*) **VALUES** (1, 'a', 1)

Если вы хотите вставить данные во все столбцы, кроме 'b', вам нужно передать столько значений, сколько столбцов вы указали в скобках:

**INSERT** **INTO** insert\_select\_testtable (\* **EXCEPT**(b)) **Values** (2, 2)

**SELECT** \* **FROM** insert\_select\_testtable

┌─a─┬─b─┬─c─┐

│ 2 │ │ 2 │

└───┴───┴───┘

┌─a─┬─b─┬─c─┐

│ 1 │ a │ 1 │

└───┴───┴───┘

В этом примере мы видим, что вторая строка содержит столбцы a и c, заполненные переданными значениями и b, заполненный значением по умолчанию.

Если список столбцов не включает все существующие столбцы, то все остальные столбцы заполняются следующим образом:

* Значения, вычисляемые из DEFAULT выражений, указанных в определении таблицы.
* Нули и пустые строки, если DEFAULT не определены.

В INSERT можно передавать данные любого [формата](https://clickhouse.tech/docs/ru/interfaces/formats/#formats), который поддерживает RT.WideStore. Для этого формат необходимо указать в запросе в явном виде:

**INSERT** **INTO** [db.]**table** [(c1, c2, c3)] FORMAT format\_name data\_set

Например, следующий формат запроса идентичен базовому варианту INSERT … VALUES:

**INSERT** **INTO** [db.]**table** [(c1, c2, c3)] FORMAT **Values** (v11, v12, v13), (v21, v22, v23), ...

RT.WideStore отсекает все пробелы и один перенос строки (если он есть) перед данными. Рекомендуем при формировании запроса переносить данные на новую строку после операторов запроса (это важно, если данные начинаются с пробелов).

Пример:

**INSERT** **INTO** t FORMAT TabSeparated

11 Hello, world!

22 Qwerty

С помощью консольного клиента или HTTP интерфейса можно вставлять данные отдельно от запроса. Как это сделать, читайте в разделе «[Интерфейсы](https://clickhouse.tech/docs/ru/interfaces/#interfaces)».

###### Ограничения (constraints)

Если в таблице объявлены [ограничения](https://clickhouse.tech/docs/ru/sql-reference/statements/create/table/#constraints), то их выполнимость будет проверена для каждой вставляемой строки. Если для хотя бы одной строки ограничения не будут выполнены, запрос будет остановлен.

###### Вставка результатов SELECT

**INSERT** **INTO** [db.]**table** [(c1, c2, c3)] **SELECT** ...

Соответствие столбцов определяется их позицией в секции SELECT. При этом, их имена в выражении SELECT и в таблице для INSERT, могут отличаться. При необходимости выполняется приведение типов данных, эквивалентное соответствующему оператору CAST.

Все форматы данных кроме Values не позволяют использовать в качестве значений выражения, такие как now(), 1 + 2 и подобные. Формат Values позволяет ограниченно использовать выражения, но это не рекомендуется, так как в этом случае для их выполнения используется неэффективный вариант кода.

Не поддерживаются другие запросы на модификацию части данных: UPDATE, DELETE, REPLACE, MERGE, UPSERT, INSERT UPDATE.

Вы можете удалять старые данные с помощью запроса ALTER TABLE ... DROP PARTITION.

Для табличной функции [input()](https://clickhouse.tech/docs/ru/sql-reference/table-functions/input/) после секции SELECT должна следовать секция FORMAT.

###### Замечания о производительности

INSERT сортирует входящие данные по первичному ключу и разбивает их на партиции по ключу партиционирования. Если вы вставляете данные в несколько партиций одновременно, то это может значительно снизить производительность запроса INSERT. Чтобы избежать этого:

* Добавляйте данные достаточно большими пачками. Например, по 100 000 строк.
* Группируйте данные по ключу партиционирования самостоятельно перед загрузкой в RT.WideStore.

Снижения производительности не будет, если:

* Данные поступают в режиме реального времени.
* Вы загружаете данные, которые как правило отсортированы по времени.

##### Запросы CREATE

Запрос create создает новую сущность одного из следующих типов:

* [DATABASE](https://clickhouse.tech/docs/ru/sql-reference/statements/create/database/),
* [TABLE](https://clickhouse.tech/docs/ru/sql-reference/statements/create/table/),
* [VIEW](https://clickhouse.tech/docs/ru/sql-reference/statements/create/view/),
* [DICTIONARY](https://clickhouse.tech/docs/ru/sql-reference/statements/create/dictionary/),
* [USER](https://clickhouse.tech/docs/ru/sql-reference/statements/create/user/),
* [ROLE](https://clickhouse.tech/docs/ru/sql-reference/statements/create/role/),
* [ROW POLICY](https://clickhouse.tech/docs/ru/sql-reference/statements/create/row-policy/),
* [QUOTA](https://clickhouse.tech/docs/ru/sql-reference/statements/create/quota/),
* [SETTINGS PROFILE](https://clickhouse.tech/docs/ru/sql-reference/statements/create/settings-profile/).

##### ALTER

Изменение структуры таблицы.

**ALTER** **TABLE** [db].name [**ON** **CLUSTER** **cluster**] **ADD**|**DROP**|CLEAR|**COMMENT**|**MODIFY** **COLUMN** ...

В запросе указывается список из одного или более действий через запятую.

Каждое действие – операция над столбцом.

Большинство запросов ALTER изменяют настройки таблицы или данные:

* [COLUMN](https://clickhouse.tech/docs/ru/sql-reference/statements/alter/column/),
* [PARTITION](https://clickhouse.tech/docs/ru/sql-reference/statements/alter/partition/),
* [DELETE](https://clickhouse.tech/docs/ru/sql-reference/statements/alter/delete/),
* [UPDATE](https://clickhouse.tech/docs/ru/sql-reference/statements/alter/update/),
* [ORDER BY](https://clickhouse.tech/docs/ru/sql-reference/statements/alter/order-by/),
* [INDEX](https://clickhouse.tech/docs/ru/sql-reference/statements/alter/index/),
* [CONSTRAINT](https://clickhouse.tech/docs/ru/sql-reference/statements/alter/constraint/),
* [TTL](https://clickhouse.tech/docs/ru/sql-reference/statements/alter/ttl/).

***Предупреждене:*** *Запрос ALTER поддерживается только для таблиц типа \*MergeTree, а также Merge и Distributed. Запрос имеет несколько вариантов.*

Следующие запросы ALTER изменяют сущности, связанные с управлением доступом на основе ролей:

* [USER](https://clickhouse.tech/docs/ru/sql-reference/statements/alter/user/),
* [ROLE](https://clickhouse.tech/docs/ru/sql-reference/statements/alter/role/),
* [QUOTA](https://clickhouse.tech/docs/ru/sql-reference/statements/alter/quota/),
* [ROW POLICY](https://clickhouse.tech/docs/ru/sql-reference/statements/alter/row-policy/),
* [SETTINGS PROFILE](https://clickhouse.tech/docs/ru/sql-reference/statements/alter/settings-profile/).

###### Мутации

Мутации – разновидность запроса ALTER, позволяющая изменять или удалять данные в таблице. В отличие от стандартных запросов [ALTER TABLE … DELETE](https://clickhouse.tech/docs/ru/sql-reference/statements/alter/delete/) и [ALTER TABLE … UPDATE](https://clickhouse.tech/docs/ru/sql-reference/statements/alter/update/), рассчитанных на точечное изменение данных, область применения мутаций – достаточно тяжёлые изменения, затрагивающие много строк в таблице. Поддержана для движков таблиц семейства [MergeTree](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/), в том числе для движков с репликацией.

Конвертировать существующие таблицы для работы с мутациями не нужно. Но после применения первой мутации формат данных таблицы становится несовместимым с предыдущими версиями и откатиться на предыдущую версию уже не получится.

На данный момент доступны команды:

**ALTER** **TABLE** [db.]**table** MATERIALIZE **INDEX** name **IN** PARTITION partition\_name

Команда перестроит вторичный индекс name для партиции partition\_name.

В одном запросе можно указать несколько команд через запятую.

Для \*MergeTree-таблиц мутации выполняются, перезаписывая данные по кускам (parts). При этом атомарности нет – куски заменяются на помутированные по мере выполнения и запрос SELECT, заданный во время выполнения мутации, увидит данные как из измененных кусков, так и из кусков, которые еще не были изменены.

Мутации линейно упорядочены между собой и накладываются на каждый кусок в порядке добавления. Мутации также упорядочены со вставками - гарантируется, что данные, вставленные в таблицу до начала выполнения запроса мутации, будут изменены, а данные, вставленные после окончания запроса мутации, изменены не будут. При этом мутации никак не блокируют вставки.

Запрос завершается немедленно после добавления информации о мутации (для реплицированных таблиц - в ZooKeeper, для нереплицированных - на файловую систему). Сама мутация выполняется асинхронно, используя настройки системного профиля. Следить за ходом её выполнения можно по таблице [system.mutations](https://clickhouse.tech/docs/ru/operations/system-tables/mutations/#system_tables-mutations). Добавленные мутации будут выполняться до конца даже в случае перезапуска серверов RT.WideStore. Откатить мутацию после её добавления нельзя, но если мутация по какой-то причине не может выполниться до конца, её можно остановить с помощью запроса [KILL MUTATION](https://clickhouse.tech/docs/ru/sql-reference/statements/kill/#kill-mutation).

Записи о последних выполненных мутациях удаляются не сразу (количество сохраняемых мутаций определяется параметром движка таблиц finished\_mutations\_to\_keep). Более старые записи удаляются.

###### Синхронность запросов ALTER

Для нереплицируемых таблиц, все запросы ALTER выполняются синхронно. Для реплицируемых таблиц, запрос всего лишь добавляет инструкцию по соответствующим действиям в ZooKeeper, а сами действия осуществляются при первой возможности. Но при этом, запрос может ждать завершения выполнения этих действий на всех репликах.

Для запросов ALTER ... ATTACH|DETACH|DROP можно настроить ожидание, с помощью настройки replication\_alter\_partitions\_sync.

Возможные значения: 0 - не ждать, 1 - ждать выполнения только у себя (по умолчанию), 2 - ждать всех.

Для запросов ALTER TABLE ... UPDATE|DELETE синхронность выполнения определяется настройкой [mutations\_sync](https://clickhouse.tech/docs/ru/operations/settings/settings/#mutations_sync).

##### Запросы SYSTEM:

* [RELOAD EMBEDDED DICTIONARIES](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query_language-system-reload-emdedded-dictionaries),
* [RELOAD DICTIONARIES](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query_language-system-reload-dictionaries),
* [RELOAD DICTIONARY](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query_language-system-reload-dictionary),
* [DROP DNS CACHE](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query_language-system-drop-dns-cache),
* [DROP MARK CACHE](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query_language-system-drop-mark-cache),
* [DROP UNCOMPRESSED CACHE](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query_language-system-drop-uncompressed-cache),
* [DROP COMPILED EXPRESSION CACHE](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query_language-system-drop-compiled-expression-cache),
* [DROP REPLICA](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query_language-system-drop-replica),
* [FLUSH LOGS](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query_language-system-flush_logs),
* [RELOAD CONFIG](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query_language-system-reload-config),
* [SHUTDOWN](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query_language-system-shutdown),
* [KILL](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query_language-system-kill),
* [STOP DISTRIBUTED SENDS](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query_language-system-stop-distributed-sends),
* [FLUSH DISTRIBUTED](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query_language-system-flush-distributed),
* [START DISTRIBUTED SENDS](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query_language-system-start-distributed-sends),
* [STOP MERGES](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query_language-system-stop-merges),
* [START MERGES](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query_language-system-start-merges),
* [STOP TTL MERGES](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query_language-stop-ttl-merges),
* [START TTL MERGES](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query_language-start-ttl-merges),
* [STOP MOVES](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query_language-stop-moves),
* [START MOVES](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query_language-start-moves),
* [STOP FETCHES](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query_language-system-stop-fetches),
* [START FETCHES](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query_language-system-start-fetches),
* [STOP REPLICATED SENDS](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query_language-system-start-replicated-sends),
* [START REPLICATED SENDS](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query_language-system-start-replicated-sends),
* [STOP REPLICATION QUEUES](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query_language-system-stop-replication-queues),
* [START REPLICATION QUEUES](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query_language-system-start-replication-queues),
* [SYNC REPLICA](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query_language-system-sync-replica),
* [RESTART REPLICA](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query_language-system-restart-replica),
* [RESTART REPLICAS](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query_language-system-restart-replicas).

###### RELOAD EMBEDDED DICTIONARIES

Перегружет все [Встроенные словари](https://clickhouse.tech/docs/ru/sql-reference/dictionaries/internal-dicts/).

По умолчанию встроенные словари выключены.

Всегда возвращает Ok., вне зависимости от результата обновления встроенных словарей.

###### RELOAD DICTIONARIES

Перегружает все словари, которые были успешно загружены до этого.

По умолчанию включена ленивая загрузка [dictionaries\_lazy\_load](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/#server_configuration_parameters-dictionaries_lazy_load), поэтому словари не загружаются автоматически при старте, а только при первом обращении через dictGet или SELECT к ENGINE=Dictionary. После этого такие словари (LOADED) будут перегружаться командой system reload dictionaries.

Всегда возвращает Ok., вне зависимости от результата обновления словарей.

###### RELOAD DICTIONARY Dictionary\_name

Полностью перегружает словарь dictionary\_name, вне зависимости от состояния словаря (LOADED/NOT\_LOADED/FAILED).

Всегда возвращает Ok., вне зависимости от результата обновления словаря.

Состояние словаря можно проверить запросом к system.dictionaries.

**SELECT** name, status **FROM** **system**.dictionaries;

###### DROP DNS CACHE

Сбрасывает внутренний DNS кеш RT.WideStore. Иногда (для старых версий RT.WideStore) необходимо использовать эту команду при изменении инфраструктуры (смене IP адреса у другого RT.WideStore сервера или сервера, используемого словарями).

Для более удобного (автоматического) управления кешем см. параметры disable\_internal\_dns\_cache, dns\_cache\_update\_period.

###### DROP MARK CACHE

Сбрасывает кеш «засечек» (mark cache). Используется при разработке RT.WideStore и тестах производительности.

###### DROP REPLICA

Мертвые реплики можно удалить, используя следующий синтаксис:

**SYSTEM** **DROP** REPLICA 'replica\_name' **FROM** **TABLE** **database**.**table**;

**SYSTEM** **DROP** REPLICA 'replica\_name' **FROM** **DATABASE** **database**;

**SYSTEM** **DROP** REPLICA 'replica\_name';

**SYSTEM** **DROP** REPLICA 'replica\_name' **FROM** ZKPATH '/path/to/table/in/zk';

Удаляет путь реплики из ZooKeeper-а. Это полезно, когда реплика мертва и ее метаданные не могут быть удалены из ZooKeeper с помощью DROP TABLE, потому что такой таблицы больше нет. DROP REPLICA может удалить только неактивную / устаревшую реплику и не может удалить локальную реплику, используйте для этого DROP TABLE. DROP REPLICA не удаляет таблицы и не удаляет данные или метаданные с диска.

Первая команда удаляет метаданные реплики 'replica\_name' для таблицы database.table.

Вторая команда удаляет метаданные реплики 'replica\_name' для всех таблиц базы данных database.

Третья команда удаляет метаданные реплики 'replica\_name' для всех таблиц, существующих на локальном сервере (список таблиц генерируется из локальной реплики).

Четверая команда полезна для удаления метаданных мертвой реплики, когда все другие реплики таблицы уже были удалены ранее, поэтому необходимо явно указать ZooKeeper путь таблицы. ZooKeeper путь – это первый аргумент для ReplicatedMergeTree движка при создании таблицы.

###### DROP UNCOMPRESSED CACHE

Сбрасывает кеш не сжатых данных. Используется при разработке RT.WideStore и тестах производительности.

Для управления кешем не сжатых данных используйте следующие настройки уровня сервера [uncompressed\_cache\_size](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/#server-settings-uncompressed_cache_size) и настройки уровня запрос/пользователь/профиль [use\_uncompressed\_cache](https://clickhouse.tech/docs/ru/operations/settings/settings/#setting-use_uncompressed_cache)

###### DROP COMPILED EXPRESSION CACHE

Сбрасывает кеш скомпилированных выражений. Используется при разработке RT.WideStore и тестах производительности.

Компилированные выражения используются когда включена настройка уровня запрос/пользователь/профиль [compile](https://clickhouse.tech/docs/ru/operations/settings/settings/#compile)

###### FLUSH LOGS

Записывает буферы логов в системные таблицы (например system.query\_log). Позволяет не ждать 7.5 секунд при отладке.

Если буфер логов пустой, то этот запрос просто создаст системные таблицы.

###### RELOAD CONFIG

Перечитывает конфигурацию настроек RT.WideStore. Используется при хранении конфигурации в zookeeeper.

###### SHUTDOWN

Штатно завершает работу RT.WideStore (аналог service clickhouse-server stop / kill {$pid\_clickhouse-server})

##### KILL

Аварийно завершает работу RT.WideStore (аналог kill -9 {$pid\_clickhouse-server})

###### Управление распределёнными таблицами

RT.WideStore может оперировать [распределёнными](https://clickhouse.tech/docs/ru/sql-reference/statements/system/) таблицами. Когда пользователь вставляет данные в эти таблицы, RT.WideStore сначала формирует очередь из данных, которые должны быть отправлены на узлы кластера, а затем асинхронно отправляет подготовленные данные. Вы можете управлять очередью с помощью запросов [STOP DISTRIBUTED SENDS](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query_language-system-stop-distributed-sends), [START DISTRIBUTED SENDS](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query_language-system-start-distributed-sends) и [FLUSH DISTRIBUTED](https://clickhouse.tech/docs/ru/sql-reference/statements/system/#query_language-system-flush-distributed). Также есть возможность синхронно вставлять распределенные данные с помощью настройки [insert\_distributed\_sync](https://clickhouse.tech/docs/ru/operations/settings/settings/#insert_distributed_sync).

STOP DISTRIBUTED SENDS

Отключает фоновую отправку при вставке данных в распределённые таблицы.

**SYSTEM** STOP DISTRIBUTED SENDS [db.]<distributed\_table\_name>

FLUSH DISTRIBUTED

В синхронном режиме отправляет все данные на узлы кластера. Если какие-либо узлы недоступны, RT.WideStore генерирует исключение и останавливает выполнение запроса. Такой запрос можно повторять до успешного завершения, что будет означать возвращение связанности с остальными узлами кластера.

**SYSTEM** FLUSH DISTRIBUTED [db.]<distributed\_table\_name>

START DISTRIBUTED SENDS

Включает фоновую отправку при вставке данных в распределенные таблицы.

**SYSTEM** **START** DISTRIBUTED SENDS [db.]<distributed\_table\_name>

Managing MergeTree Tables

RT.WideStore может управлять фоновыми процессами в [MergeTree](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/) таблицах.

STOP MERGES

Позволяет остановить фоновые мержи для таблиц семейства MergeTree:

**SYSTEM** STOP MERGES [**ON** VOLUME <volume\_name> | [db.]merge\_tree\_family\_table\_name]

Note

DETACH / ATTACH таблицы восстанавливает фоновые мержи для этой таблицы (даже в случае отключения фоновых мержей для всех таблиц семейства MergeTree до DETACH).

START MERGES

Включает фоновые мержи для таблиц семейства MergeTree:

**SYSTEM** **START** MERGES [**ON** VOLUME <volume\_name> | [db.]merge\_tree\_family\_table\_name]

STOP TTL MERGES

Позволяет остановить фоновые процессы удаления старых данных основанные на [выражениях TTL](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/#table_engine-mergetree-ttl) для таблиц семейства MergeTree:

Возвращает Ok. даже если указана несуществующая таблица или таблица имеет тип отличный от MergeTree. Возвращает ошибку если указана не существующая база данных:

**SYSTEM** STOP TTL MERGES [[db.]merge\_tree\_family\_table\_name]

START TTL MERGES

Запускает фоновые процессы удаления старых данных основанные на [выражениях TTL](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/#table_engine-mergetree-ttl) для таблиц семейства MergeTree:

Возвращает Ok. даже если указана несуществующая таблица или таблица имеет тип отличный от MergeTree. Возвращает ошибку если указана не существующая база данных:

**SYSTEM** **START** TTL MERGES [[db.]merge\_tree\_family\_table\_name]

STOP MOVES

Позволяет остановить фоновые процессы переноса данных основанные [табличных выражениях TTL с использованием TO VOLUME или TO DISK](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/#mergetree-table-ttl) for tables in the MergeTree family:

Возвращает Ok. даже если указана несуществующая таблица или таблица имеет тип отличный от MergeTree. Возвращает ошибку если указана не существующая база данных:

**SYSTEM** STOP MOVES [[db.]merge\_tree\_family\_table\_name]

START MOVES

Запускает фоновые процессы переноса данных основанные [табличных выражениях TTL с использованием TO VOLUME или TO DISK](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/#mergetree-table-ttl) for tables in the MergeTree family:

Возвращает Ok. даже если указана несуществующая таблица или таблица имеет тип отличный от MergeTree. Возвращает ошибку если указана не существующая база данных:

**SYSTEM** STOP MOVES [[db.]merge\_tree\_family\_table\_name]

Managing ReplicatedMergeTree Tables

RT.WideStore может управлять фоновыми процессами связанными c репликацией в таблицах семейства [ReplicatedMergeTree](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/replacingmergetree/).

STOP FETCHES

Позволяет остановить фоновые процессы синхронизации новыми вставленными кусками данных с другими репликами в кластере для таблиц семейства ReplicatedMergeTree:

Всегда возвращает Ok. вне зависимости от типа таблицы и даже если таблица или база данных не существет:

**SYSTEM** STOP FETCHES [[db.]replicated\_merge\_tree\_family\_table\_name]

START FETCHES

Позволяет запустить фоновые процессы синхронизации новыми вставленными кусками данных с другими репликами в кластере для таблиц семейства ReplicatedMergeTree:

Всегда возвращает Ok. вне зависимости от типа таблицы и даже если таблица или база данных не существет:

**SYSTEM** **START** FETCHES [[db.]replicated\_merge\_tree\_family\_table\_name]

STOP REPLICATED SENDS

Позволяет остановить фоновые процессы отсылки новых вставленных кусков данных другим репликам в кластере для таблиц семейства ReplicatedMergeTree:

**SYSTEM** STOP REPLICATED SENDS [[db.]replicated\_merge\_tree\_family\_table\_name]

START REPLICATED SENDS

Позволяет запустить фоновые процессы отсылки новых вставленных кусков данных другим репликам в кластере для таблиц семейства ReplicatedMergeTree:

**SYSTEM** **START** REPLICATED SENDS [[db.]replicated\_merge\_tree\_family\_table\_name]

STOP REPLICATION QUEUES

Останавливает фоновые процессы разбора заданий из очереди репликации, которая хранится в Zookeeper для таблиц семейства ReplicatedMergeTree. Возможные типы заданий - merges, fetches, mutation, DDL запросы с ON CLUSTER:

**SYSTEM** STOP REPLICATION QUEUES [[db.]replicated\_merge\_tree\_family\_table\_name]

START REPLICATION QUEUES

Запускает фоновые процессы разбора заданий из очереди репликации, которая хранится в Zookeeper для таблиц семейства ReplicatedMergeTree. Возможные типы заданий - merges, fetches, mutation, DDL запросы с ON CLUSTER:

**SYSTEM** **START** REPLICATION QUEUES [[db.]replicated\_merge\_tree\_family\_table\_name]

SYNC REPLICA

Ждет, когда таблица семейства ReplicatedMergeTree будет синхронизирована с другими репликами в кластере, будет работать до достижения receive\_timeout, если синхронизация для таблицы отключена в настоящий момент времени:

**SYSTEM** SYNC REPLICA [db.]replicated\_merge\_tree\_family\_table\_name

RESTART REPLICA

Реинициализация состояния Zookeeper сессий для таблицы семейства ReplicatedMergeTree, сравнивает текущее состояние с тем что хранится в Zookeeper как источник правды и добавляет задачи Zookeeper очередь если необходимо

Инициализация очереди репликации на основе данных ZooKeeper, происходит так же как при attach table. На короткое время таблица станет недоступной для любых операций:

**SYSTEM** **RESTART** REPLICA [db.]replicated\_merge\_tree\_family\_table\_name

RESTART REPLICAS

Реинициализация состояния Zookeeper сессий для всех ReplicatedMergeTree таблиц, сравнивает текущее состояние с тем что хранится в Zookeeper как источник правды и добавляет задачи Zookeeper очередь если необходимо

##### SHOW Queries

###### SHOW CREATE TABLE

**SHOW** **CREATE** [**TEMPORARY**] [**TABLE**|**DICTIONARY**] [db.]**table** [**INTO** OUTFILE filename] [FORMAT format]

Возвращает один столбец типа String с именем statement, содержащий одно значение – запрос CREATE TABLE, с помощью которого был создан указанный объект.

###### SHOW DATABASES

Выводит список всех баз данных:

**SHOW** DATABASES [**LIKE** | **ILIKE** | **NOT** **LIKE** '<pattern>'] [**LIMIT** <N>] [**INTO** OUTFILE filename] [FORMAT format]

Этот запрос идентичен запросу:

**SELECT** name **FROM** **system**.databases [**WHERE** name **LIKE** | **ILIKE** | **NOT** **LIKE** '<pattern>'] [**LIMIT** <N>] [**INTO** OUTFILE filename] [FORMAT format]

Примеры

Получение списка баз данных, имена которых содержат последовательность символов 'de':

**SHOW** DATABASES **LIKE** '%de%'

Результат:

┌─name────┐

│ default │

└─────────┘

Получение списка баз данных, имена которых содержат последовательность символов 'de' независимо от регистра:

SHOW DATABASES ILIKE '%DE%'

Результат:

┌─name────┐

│ default │

└─────────┘

Получение списка баз данных, имена которых не содержат последовательность символов 'de':

**SHOW** DATABASES **NOT** **LIKE** '%de%'

Результат:

┌─name───────────────────────────┐

│ \_temporary\_and\_external\_tables │

│ system │

│ test │

│ tutorial │

└────────────────────────────────┘

Получение первых двух строк из списка имен баз данных:

**SHOW** DATABASES **LIMIT** 2

Результат:

┌─name───────────────────────────┐

│ \_temporary\_and\_external\_tables │

│ default │

└────────────────────────────────┘

Смотрите также

* [CREATE DATABASE](https://clickhouse.tech/docs/ru/sql-reference/statements/create/database/#query-language-create-database).

###### SHOW PROCESSLIST

**SHOW** PROCESSLIST [**INTO** OUTFILE filename] [FORMAT format]

Выводит содержимое таблицы [system.processes](https://clickhouse.tech/docs/ru/operations/system-tables/processes/#system_tables-processes), которая содержит список запросов, выполняющихся в данный момент времени, кроме самих запросов SHOW PROCESSLIST.

Запрос SELECT \* FROM system.processes возвращает данные обо всех текущих запросах.

Полезный совет (выполните в консоли):

$ watch -n1 "clickhouse-client --query='SHOW PROCESSLIST'"

###### SHOW TABLES

Выводит список таблиц:

**SHOW** [**TEMPORARY**] TABLES [{**FROM** | **IN**} <db>] [**LIKE** | **ILIKE** | **NOT** **LIKE** '<pattern>'] [**LIMIT** <N>] [**INTO** OUTFILE <filename>] [FORMAT <format>]

Если условие FROM не указано, запрос возвращает список таблиц из текущей базы данных.

Этот запрос идентичен запросу:

**SELECT** name **FROM** **system**.tables [**WHERE** name **LIKE** | **ILIKE** | **NOT** **LIKE** '<pattern>'] [**LIMIT** <N>] [**INTO** OUTFILE <filename>] [FORMAT <format>]

Примеры

Получение списка таблиц, имена которых содержат последовательность символов 'user':

**SHOW** TABLES **FROM** **system** **LIKE** '%user%'

Результат:

┌─name─────────────┐

│ user\_directories │

│ users │

└──────────────────┘

Получение списка таблиц, имена которых содержат последовательность символов 'user' без учета регистра:

**SHOW** TABLES **FROM** **system** **ILIKE** '%USER%'

Результат:

┌─name─────────────┐

│ user\_directories │

│ users │

└──────────────────┘

Получение списка таблиц, имена которых не содержат символ 's':

**SHOW** TABLES **FROM** **system** **NOT** **LIKE** '%s%'

Результат:

┌─name─────────┐

│ metric\_log │

│ metric\_log\_0 │

│ metric\_log\_1 │

└──────────────┘

Получение первых двух строк из списка таблиц:

**SHOW** TABLES **FROM** **system** **LIMIT** 2

Результат:

┌─name───────────────────────────┐

│ aggregate\_function\_combinators │

│ asynchronous\_metric\_log │

└────────────────────────────────┘

Смотрите также

* [Create Tables](https://clickhouse.tech/docs/ru/getting-started/tutorial/#create-tables),
* [SHOW CREATE TABLE](https://clickhouse.tech/docs/ru/sql-reference/statements/show/#show-create-table).

###### SHOW DICTIONARIES

Выводит список [внешних словарей](https://clickhouse.tech/docs/ru/sql-reference/statements/show/):

**SHOW** DICTIONARIES [**FROM** <db>] [**LIKE** '<pattern>'] [**LIMIT** <N>] [**INTO** OUTFILE <filename>] [FORMAT <format>]

Если секция FROM не указана, запрос возвращает список словарей из текущей базы данных.

Аналогичный результат можно получить следующим запросом:

**SELECT** name **FROM** **system**.dictionaries **WHERE** **database** = <db> [**AND** name **LIKE** <pattern>] [**LIMIT** <N>] [**INTO** OUTFILE <filename>] [FORMAT <format>]

Пример:

Запрос выводит первые две стоки из списка таблиц в базе данных system, имена которых содержат reg:

**SHOW** DICTIONARIES **FROM** db **LIKE** '%reg%' **LIMIT** 2

┌─name─────────┐

│ regions │

│ region\_names │

└──────────────┘

###### SHOW GRANTS

Выводит привилегии пользователя.

Синтаксис:

**SHOW** GRANTS [**FOR** **user**]

Если пользователь не задан, запрос возвращает привилегии текущего пользователя.

###### SHOW CREATE USER

Выводит параметры, использованные при [создании пользователя](https://clickhouse.tech/docs/ru/sql-reference/statements/create/user/#create-user-statement).

SHOW CREATE USER не возвращает пароль пользователя.

Синтаксис:

**SHOW** **CREATE** **USER** [name1 [, name2 ...] | **CURRENT\_USER**]

###### SHOW CREATE ROLE

Выводит параметры, использованные при [создании роли](https://clickhouse.tech/docs/ru/sql-reference/statements/create/role/#create-role-statement).

Синтаксис:

**SHOW** **CREATE** **ROLE** name1 [, name2 ...]

###### SHOW CREATE ROW POLICY

Выводит параметры, использованные при [создании политики доступа к строкам](https://clickhouse.tech/docs/ru/sql-reference/statements/create/row-policy/#create-row-policy-statement).

Синтаксис:

**SHOW** **CREATE** [**ROW**] POLICY name **ON** [database1.]table1 [, [database2.]table2 ...]

###### SHOW CREATE QUOTA

Выводит параметры, использованные при [создании квоты](https://clickhouse.tech/docs/ru/sql-reference/statements/create/quota/#create-quota-statement).

Синтаксис:

**SHOW** **CREATE** QUOTA [name1 [, name2 ...] | **CURRENT**]

###### SHOW CREATE SETTINGS PROFILE

Выводит параметры, использованные при [создании профиля настроек](https://clickhouse.tech/docs/ru/sql-reference/statements/create/settings-profile/#create-settings-profile-statement).

Синтаксис:

**SHOW** **CREATE** [SETTINGS] PROFILE name1 [, name2 ...]

###### SHOW USERS

Выводит список [пользовательских аккаунтов](https://clickhouse.tech/docs/ru/operations/access-rights/#user-account-management). Для просмотра параметров пользовательских аккаунтов, см. системную таблицу [system.users](https://clickhouse.tech/docs/ru/operations/system-tables/users/#system_tables-users).

Синтаксис:

**SHOW** USERS

###### SHOW ROLES

Выводит список [ролей](https://clickhouse.tech/docs/ru/operations/access-rights/#role-management). Для просмотра параметров ролей, см. системные таблицы [system.roles](https://clickhouse.tech/docs/ru/operations/system-tables/roles/#system_tables-roles) и [system.role-grants](https://clickhouse.tech/docs/ru/operations/system-tables/role-grants/#system_tables-role_grants).

Синтаксис:

**SHOW** [**CURRENT**|ENABLED] ROLES

###### SHOW PROFILES

Выводит список [профилей настроек](https://clickhouse.tech/docs/ru/operations/access-rights/#settings-profiles-management). Для просмотра других параметров профилей настроек, см. системную таблицу [settings\_profiles](https://clickhouse.tech/docs/ru/operations/system-tables/settings_profiles/#system_tables-settings_profiles).

Синтаксис:

**SHOW** [SETTINGS] PROFILES

###### SHOW POLICIES

Выводит список [политик доступа к строкам](https://clickhouse.tech/docs/ru/operations/access-rights/#row-policy-management) для указанной таблицы. Для просмотра других параметров, см. системную таблицу [system.row\_policies](https://clickhouse.tech/docs/ru/operations/system-tables/row_policies/#system_tables-row_policies).

Синтаксис:

**SHOW** [**ROW**] POLICIES [**ON** [db.]**table**]

###### SHOW QUOTAS

Выводит список [квот](https://clickhouse.tech/docs/ru/operations/access-rights/#quotas-management). Для просмотра параметров квот, см. системную таблицу [system.quotas](https://clickhouse.tech/docs/ru/operations/system-tables/quotas/#system_tables-quotas).

Синтаксис:

**SHOW** QUOTAS

###### SHOW QUOTA

Выводит потребление [квоты](https://clickhouse.tech/docs/ru/operations/quotas/) для всех пользователей или только для текущего пользователя. Для просмотра других параметров, см. системные таблицы [system.quotas\_usage](https://clickhouse.tech/docs/ru/operations/system-tables/quotas_usage/#system_tables-quotas_usage) и [system.quota\_usage](https://clickhouse.tech/docs/ru/operations/system-tables/quota_usage/#system_tables-quota_usage).

Синтаксис:

**SHOW** [**CURRENT**] QUOTA

###### SHOW ACCESS

Выводит список всех [пользователей](https://clickhouse.tech/docs/ru/operations/access-rights/#user-account-management), [ролей](https://clickhouse.tech/docs/ru/operations/access-rights/#role-management), [профилей](https://clickhouse.tech/docs/ru/operations/access-rights/#settings-profiles-management) и пр., а также все [привилегии](https://clickhouse.tech/docs/ru/sql-reference/statements/grant/#grant-privileges).

Синтаксис:

**SHOW** **ACCESS**

##### GRANT

* Присваивает [привилегии](https://clickhouse.tech/docs/ru/sql-reference/statements/grant/#grant-privileges) пользователям или ролям RT.WideStore.
* Назначает роли пользователям или другим ролям.

Отозвать привилегию можно с помощью выражения [REVOKE](https://clickhouse.tech/docs/ru/sql-reference/statements/revoke/). Чтобы вывести список присвоенных привилегий, воспользуйтесь выражением [SHOW GRANTS](https://clickhouse.tech/docs/ru/sql-reference/statements/show/#show-grants-statement).

###### Синтаксис присвоения привилегий

**GRANT** [**ON** **CLUSTER** cluster\_name] privilege[(**column\_name** [,...])] [,...] **ON** {db.**table**|db.\*|\*.\*|**table**|\*} **TO** {**user** | **role** | **CURRENT\_USER**} [,...] [**WITH** **GRANT** **OPTION**]

* privilege – тип привилегии
* role – роль пользователя RT.WideStore.
* user – пользователь RT.WideStore.

WITH GRANT OPTION разрешает пользователю или роли выполнять запрос GRANT. Пользователь может выдавать только те привилегии, которые есть у него, той же или меньшей области действий.

Синтаксис назначения ролей:

**GRANT** [**ON** **CLUSTER** cluster\_name] **role** [,...] **TO** {**user** | another\_role | **CURRENT\_USER**} [,...] [**WITH** **ADMIN** **OPTION**]

* role – роль пользователя RT.WideStore.
* user – пользователь RT.WideStore.

WITH ADMIN OPTION присваивает привилегию [ADMIN OPTION](https://clickhouse.tech/docs/ru/sql-reference/statements/grant/#admin-option-privilege) пользователю или роли.

###### Использование

Для использования GRANT пользователь должен иметь привилегию GRANT OPTION. Пользователь может выдавать привилегии только внутри области действий, назначенных ему самому привилегий.

Например, администратор выдал привилегию пользователю john:

**GRANT** **SELECT**(x,y) **ON** db.**table** **TO** john **WITH** **GRANT** **OPTION**

Это означает, что пользователю john разрешено выполнять:

* SELECT x,y FROM db.table.
* SELECT x FROM db.table.
* SELECT y FROM db.table.

john не может выполнить SELECT z FROM db.table или SELECT \* FROM db.table. После обработки данных запросов RT.WideStore ничего не вернет – даже x или y. Единственное исключение – если таблица содержит только столбцы x и y. В таком случае RT.WideStore вернет все данные.

Также у john есть привилегия GRANT OPTION. john может выдать другим пользователям привилегии той же или меньшей области действий из тех, которые есть у него.

При присвоении привилегий допускается использовать астериск (\*) вместо имени таблицы или базы данных. Например, запрос GRANT SELECT ON db.\* TO john позволит пользователю john выполнять SELECT над всеми таблицами в базе данных db. Также вы можете опускать имя базы данных. В таком случае привилегии позволят совершать операции над текущей базой данных. Например, запрос GRANT SELECT ON \* TO john выдаст привилегию на выполнение SELECT над всеми таблицами в текущей базе данных, GRANT SELECT ON mytable TO john – только над таблицей mytable в текущей базе данных.

Доступ к базе данных system разрешен всегда (данная база данных используется при обработке запросов).

Вы можете присвоить несколько привилегий нескольким пользователям в одном запросе. Запрос GRANT SELECT, INSERT ON \*.\* TO john, robin позволит пользователям john и robin выполнять INSERT и SELECT над всеми таблицами всех баз данных на сервере.

###### Привилегии

Привилегия – это разрешение на выполнение определенного типа запросов.

Привилегии имеют иерархическую структуру. Набор разрешенных запросов зависит от области действия привилегии.

Иерархия привилегий:

* [SELECT](https://clickhouse.tech/docs/ru/sql-reference/statements/grant/#grant-select)
* [INSERT](https://clickhouse.tech/docs/ru/sql-reference/statements/grant/#grant-insert)
* [ALTER](https://clickhouse.tech/docs/ru/sql-reference/statements/grant/#grant-alter):
  + ALTER TABLE:
    - ALTER UPDATE
    - ALTER DELETE
    - ALTER COLUMN:
      * ALTER ADD COLUMN
      * ALTER DROP COLUMN
      * ALTER MODIFY COLUMN
      * ALTER COMMENT COLUMN
      * ALTER CLEAR COLUMN
      * ALTER RENAME COLUMN
    - ALTER INDEX:
      * ALTER ORDER BY
      * ALTER SAMPLE BY
      * ALTER ADD INDEX
      * ALTER DROP INDEX
      * ALTER MATERIALIZE INDEX
      * ALTER CLEAR INDEX
    - ALTER CONSTRAINT:
      * ALTER ADD CONSTRAINT
      * ALTER DROP CONSTRAINT
    - ALTER TTL
    - ALTER MATERIALIZE TTL
    - ALTER SETTINGS
    - ALTER MOVE PARTITION
    - ALTER FETCH PARTITION
    - ALTER FREEZE PARTITION
  + ALTER VIEW:
    - ALTER VIEW REFRESH
    - ALTER VIEW MODIFY QUERY
* [CREATE](https://clickhouse.tech/docs/ru/sql-reference/statements/grant/#grant-create):
  + CREATE DATABASE
  + CREATE TABLE
  + CREATE VIEW
  + CREATE DICTIONARY
  + CREATE TEMPORARY TABLE
* [DROP](https://clickhouse.tech/docs/ru/sql-reference/statements/grant/#grant-drop):
  + DROP DATABASE
  + DROP TABLE
  + DROP VIEW
  + DROP DICTIONARY
* [TRUNCATE](https://clickhouse.tech/docs/ru/sql-reference/statements/grant/#grant-truncate)
* [OPTIMIZE](https://clickhouse.tech/docs/ru/sql-reference/statements/grant/#grant-optimize)
* [SHOW](https://clickhouse.tech/docs/ru/sql-reference/statements/grant/#grant-show):
  + SHOW DATABASES
  + SHOW TABLES
  + SHOW COLUMNS
  + SHOW DICTIONARIES
* [KILL QUERY](https://clickhouse.tech/docs/ru/sql-reference/statements/grant/#grant-kill-query)
* [ACCESS MANAGEMENT](https://clickhouse.tech/docs/ru/sql-reference/statements/grant/#grant-access-management):
  + CREATE USER
  + ALTER USER
  + DROP USER
  + CREATE ROLE
  + ALTER ROLE
  + DROP ROLE
  + CREATE ROW POLICY
  + ALTER ROW POLICY
  + DROP ROW POLICY
  + CREATE QUOTA
  + ALTER QUOTA
  + DROP QUOTA
  + CREATE SETTINGS PROFILE
  + ALTER SETTINGS PROFILE
  + DROP SETTINGS PROFILE
  + SHOW ACCESS:
    - SHOW\_USERS
    - SHOW\_ROLES
    - SHOW\_ROW\_POLICIES
    - SHOW\_QUOTAS
    - SHOW\_SETTINGS\_PROFILES
  + ROLE ADMIN
* [SYSTEM](https://clickhouse.tech/docs/ru/sql-reference/statements/grant/#grant-system):
  + SYSTEM SHUTDOWN
  + SYSTEM DROP CACHE:
    - SYSTEM DROP DNS CACHE
    - SYSTEM DROP MARK CACHE
    - SYSTEM DROP UNCOMPRESSED CACHE
  + SYSTEM RELOAD:
    - SYSTEM RELOAD CONFIG
    - SYSTEM RELOAD DICTIONARY
    - SYSTEM RELOAD EMBEDDED DICTIONARIES
  + SYSTEM MERGES
  + SYSTEM TTL MERGES
  + SYSTEM FETCHES
  + SYSTEM MOVES
  + SYSTEM SENDS:
    - SYSTEM DISTRIBUTED SENDS
    - SYSTEM REPLICATED SENDS
  + SYSTEM REPLICATION QUEUES
  + SYSTEM SYNC REPLICA
  + SYSTEM RESTART REPLICA
  + SYSTEM FLUSH:
    - SYSTEM FLUSH DISTRIBUTED
    - SYSTEM FLUSH LOGS
* [INTROSPECTION](https://clickhouse.tech/docs/ru/sql-reference/statements/grant/#grant-introspection):
  + addressToLine
  + addressToSymbol
  + demangle
* [SOURCES](https://clickhouse.tech/docs/ru/sql-reference/statements/grant/#grant-sources):
  + FILE
  + URL
  + REMOTE
  + MYSQL
  + ODBC
  + JDBC
  + HDFS
  + S3
* [dictGet](https://clickhouse.tech/docs/ru/sql-reference/statements/grant/#grant-dictget)

Примеры того, как трактуется данная иерархия:

* Привилегия ALTER включает все остальные ALTER\* привилегии.
* ALTER CONSTRAINT включает ALTER ADD CONSTRAINT и ALTER DROP CONSTRAINT.

Привилегии применяются на разных уровнях. Уровень определяет синтаксис присваивания привилегии.

Уровни (от низшего к высшему):

* COLUMN – привилегия присваивается для столбца, таблицы, базы данных или глобально.
* TABLE – привилегия присваивается для таблицы, базы данных или глобально.
* VIEW – привилегия присваивается для представления, базы данных или глобально.
* DICTIONARY – привилегия присваивается для словаря, базы данных или глобально.
* DATABASE – Привилегия присваивается для базы данных или глобально.
* GLOBAL – привилегия присваивается только глобально.
* GROUP – группирует привилегии разных уровней. При присвоении привилегии уровня GROUP присваиваются только привилегии из группы в соответствии с используемым синтаксисом.

Примеры допустимого синтаксиса:

* GRANT SELECT(x) ON db.table TO user
* GRANT SELECT ON db.\* TO user

Примеры недопустимого синтаксиса:

* GRANT CREATE USER(x) ON db.table TO user
* GRANT CREATE USER ON db.\* TO user

Специальная привилегия [ALL](https://clickhouse.tech/docs/ru/sql-reference/statements/grant/#grant-all) присваивает все привилегии пользователю или роли.

По умолчанию пользователь или роль не имеют привилегий.

Отсутствие привилегий у пользователя или роли отображается как привилегия [NONE](https://clickhouse.tech/docs/ru/sql-reference/statements/grant/#grant-none).

Выполнение некоторых запросов требует определенного набора привилегий. Например, чтобы выполнить запрос [RENAME](https://clickhouse.tech/docs/ru/sql-reference/statements/misc/#misc_operations-rename), нужны следующие привилегии: SELECT, CREATE TABLE, INSERT и DROP TABLE.

SELECT

Разрешает выполнять запросы [SELECT](https://clickhouse.tech/docs/ru/sql-reference/statements/select/).

Уровень: COLUMN.

Описание:

Пользователь с данной привилегией может выполнять запросы SELECT над определенными столбцами из определенной таблицы и базы данных. При включении в запрос других столбцов запрос ничего не вернет.

Рассмотрим следующую привилегию:

**GRANT** **SELECT**(x,y) **ON** db.**table** **TO** john

Данная привилегия позволяет пользователю john выполнять выборку данных из столбцов x и/или y в db.table, например, SELECT x FROM db.table. john не может выполнить SELECT z FROM db.table или SELECT \* FROM db.table. После обработки данных запросов RT.WideStore ничего не вернет – даже x или y. Единственное исключение – если таблица содержит только столбцы x и y. В таком случае RT.WideStore вернет все данные.

INSERT

Разрешает выполнять запросы [INSERT](https://clickhouse.tech/docs/ru/sql-reference/statements/insert-into/).

Уровень: COLUMN.

Описание:

Пользователь с данной привилегией может выполнять запросы INSERT над определенными столбцами из определенной таблицы и базы данных. При включении в запрос других столбцов запрос не добавит никаких данных.

Пример:

**GRANT** **INSERT**(x,y) **ON** db.**table** **TO** john

Присвоенная привилегия позволит пользователю john вставить данные в столбцы x и/или y в db.table.

ALTER

Разрешает выполнять запросы [ALTER](https://clickhouse.tech/docs/ru/sql-reference/statements/alter/) в соответствии со следующей иерархией привилегий:

* ALTER. Уровень: COLUMN:
  + ALTER TABLE. Уровень: GROUP:
    - ALTER UPDATE. Уровень: COLUMN. Алиасы: UPDATE
    - ALTER DELETE. Уровень: COLUMN. Алиасы: DELETE
    - ALTER COLUMN. Уровень: GROUP:
      * ALTER ADD COLUMN. Уровень: COLUMN. Алиасы: ADD COLUMN
      * ALTER DROP COLUMN. Уровень: COLUMN. Алиасы: DROP COLUMN
      * ALTER MODIFY COLUMN. Уровень: COLUMN. Алиасы: MODIFY COLUMN
      * ALTER COMMENT COLUMN. Уровень: COLUMN. Алиасы: COMMENT COLUMN
      * ALTER CLEAR COLUMN. Уровень: COLUMN. Алиасы: CLEAR COLUMN
      * ALTER RENAME COLUMN. Уровень: COLUMN. Алиасы: RENAME COLUMN
    - ALTER INDEX. Уровень: GROUP. Алиасы: INDEX:
      * ALTER ORDER BY. Уровень: TABLE. Алиасы: ALTER MODIFY ORDER BY, MODIFY ORDER BY
      * ALTER SAMPLE BY. Уровень: TABLE. Алиасы: ALTER MODIFY SAMPLE BY, MODIFY SAMPLE BY
      * ALTER ADD INDEX. Уровень: TABLE. Алиасы: ADD INDEX
      * ALTER DROP INDEX. Уровень: TABLE. Алиасы: DROP INDEX
      * ALTER MATERIALIZE INDEX. Уровень: TABLE. Алиасы: MATERIALIZE INDEX
      * ALTER CLEAR INDEX. Уровень: TABLE. Алиасы: CLEAR INDEX
    - ALTER CONSTRAINT. Уровень: GROUP. Алиасы: CONSTRAINT:
      * ALTER ADD CONSTRAINT. Уровень: TABLE. Алиасы: ADD CONSTRAINT
      * ALTER DROP CONSTRAINT. Уровень: TABLE. Алиасы: DROP CONSTRAINT
    - ALTER TTL. Уровень: TABLE. Алиасы: ALTER MODIFY TTL, MODIFY TTL
    - ALTER MATERIALIZE TTL. Уровень: TABLE. Алиасы: MATERIALIZE TTL
    - ALTER SETTINGS. Уровень: TABLE. Алиасы: ALTER SETTING, ALTER MODIFY SETTING, MODIFY SETTING
    - ALTER MOVE PARTITION. Уровень: TABLE. Алиасы: ALTER MOVE PART, MOVE PARTITION, MOVE PART
    - ALTER FETCH PARTITION. Уровень: TABLE. Алиасы: FETCH PARTITION
    - ALTER FREEZE PARTITION. Уровень: TABLE. Алиасы: FREEZE PARTITION
  + ALTER VIEW Уровень: GROUP:
    - ALTER VIEW REFRESH. Уровень: VIEW. Алиасы: ALTER LIVE VIEW REFRESH, REFRESH VIEW
    - ALTER VIEW MODIFY QUERY. Уровень: VIEW. Алиасы: ALTER TABLE MODIFY QUERY

Примеры того, как трактуется данная иерархия:

* Привилегия ALTER включает все остальные ALTER\* привилегии.
* ALTER CONSTRAINT включает ALTER ADD CONSTRAINT и ALTER DROP CONSTRAINT.

Дополнительно:

* Привилегия MODIFY SETTING позволяет изменять настройки движков таблиц. Не влияет на настройки или конфигурационные параметры сервера.
* Операция ATTACH требует наличие привилегии [CREATE](https://clickhouse.tech/docs/ru/sql-reference/statements/grant/#grant-create).
* Операция DETACH требует наличие привилегии [DROP](https://clickhouse.tech/docs/ru/sql-reference/statements/grant/#grant-drop).
* Для остановки мутации с помощью [KILL MUTATION](https://clickhouse.tech/docs/ru/sql-reference/statements/kill/#kill-mutation), необходима привилегия на выполнение данной мутации. Например, чтобы остановить запрос ALTER UPDATE, необходима одна из привилегий: ALTER UPDATE, ALTER TABLE или ALTER.

CREATE

Разрешает выполнять DDL-запросы [CREATE](https://clickhouse.tech/docs/ru/sql-reference/statements/create/) и [ATTACH](https://clickhouse.tech/docs/ru/sql-reference/statements/misc/#attach) в соответствии со следующей иерархией привилегий:

* CREATE. Уровень: GROUP:
  + CREATE DATABASE. Уровень: DATABASE
  + CREATE TABLE. Уровень: TABLE
  + CREATE VIEW. Уровень: VIEW
  + CREATE DICTIONARY. Уровень: DICTIONARY
  + CREATE TEMPORARY TABLE. Уровень: GLOBAL

Дополнительно:

* Для удаления созданной таблицы пользователю необходима привилегия [DROP](https://clickhouse.tech/docs/ru/sql-reference/statements/grant/#grant-drop).

DROP

Разрешает выполнять запросы [DROP](https://clickhouse.tech/docs/ru/sql-reference/statements/misc/#drop) и [DETACH](https://clickhouse.tech/docs/ru/sql-reference/statements/misc/#detach-statement) в соответствии со следующей иерархией привилегий:

* DROP. Уровень:
  + DROP DATABASE. Уровень: DATABASE
  + DROP TABLE. Уровень: TABLE
  + DROP VIEW. Уровень: VIEW
  + DROP DICTIONARY. Уровень: DICTIONARY

TRUNCATE

Разрешает выполнять запросы [TRUNCATE](https://clickhouse.tech/docs/ru/sql-reference/statements/truncate/).

Уровень: TABLE.

OPTIMIZE

Разрешает выполнять запросы [OPTIMIZE TABLE](https://clickhouse.tech/docs/ru/sql-reference/statements/misc/#misc_operations-optimize).

Уровень: TABLE.

SHOW

Разрешает выполнять запросы SHOW, DESCRIBE, USE и EXISTS в соответствии со следующей иерархией привилегий:

* SHOW. Уровень: GROUP:
  + SHOW DATABASES. Уровень: DATABASE. Разрешает выполнять запросы SHOW DATABASES, SHOW CREATE DATABASE, USE <database>.
  + SHOW TABLES. Уровень: TABLE. Разрешает выполнять запросы SHOW TABLES, EXISTS <table>, CHECK <table>.
  + SHOW COLUMNS. Уровень: COLUMN. Разрешает выполнять запросы SHOW CREATE TABLE, DESCRIBE.
  + SHOW DICTIONARIES. Уровень: DICTIONARY. Разрешает выполнять запросы SHOW DICTIONARIES, SHOW CREATE DICTIONARY, EXISTS <dictionary>.

Дополнительно:

У пользователя есть привилегия SHOW, если ему присвоена любая другая привилегия по отношению к определенной таблице, словарю или базе данных.

KILL QUERY

Разрешает выполнять запросы [KILL](https://clickhouse.tech/docs/ru/sql-reference/statements/kill/#kill-query) в соответствии со следующей иерархией привилегий:

Уровень: GLOBAL.

Дополнительно:

KILL QUERY позволяет пользователю останавливать запросы других пользователей.

ACCESS MANAGEMENT

Разрешает пользователю выполнять запросы на управление пользователями, ролями и политиками доступа к строкам:

* ACCESS MANAGEMENT. Уровень: GROUP:
  + CREATE USER. Уровень: GLOBAL
  + ALTER USER. Уровень: GLOBAL
  + DROP USER. Уровень: GLOBAL
  + CREATE ROLE. Уровень: GLOBAL
  + ALTER ROLE. Уровень: GLOBAL
  + DROP ROLE. Уровень: GLOBAL
  + ROLE ADMIN. Уровень: GLOBAL
  + CREATE ROW POLICY. Уровень: GLOBAL. Алиасы: CREATE POLICY
  + ALTER ROW POLICY. Уровень: GLOBAL. Алиасы: ALTER POLICY
  + DROP ROW POLICY. Уровень: GLOBAL. Алиасы: DROP POLICY
  + CREATE QUOTA. Уровень: GLOBAL
  + ALTER QUOTA. Уровень: GLOBAL
  + DROP QUOTA. Уровень: GLOBAL
  + CREATE SETTINGS PROFILE. Уровень: GLOBAL. Алиасы: CREATE PROFILE
  + ALTER SETTINGS PROFILE. Уровень: GLOBAL. Алиасы: ALTER PROFILE
  + DROP SETTINGS PROFILE. Уровень: GLOBAL. Алиасы: DROP PROFILE
  + SHOW ACCESS. Уровень: GROUP:
    - SHOW\_USERS. Уровень: GLOBAL. Алиасы: SHOW CREATE USER
    - SHOW\_ROLES. Уровень: GLOBAL. Алиасы: SHOW CREATE ROLE
    - SHOW\_ROW\_POLICIES. Уровень: GLOBAL. Алиасы: SHOW POLICIES, SHOW CREATE ROW POLICY, SHOW CREATE POLICY
    - SHOW\_QUOTAS. Уровень: GLOBAL. Алиасы: SHOW CREATE QUOTA
    - SHOW\_SETTINGS\_PROFILES. Уровень: GLOBAL. Алиасы: SHOW PROFILES, SHOW CREATE SETTINGS PROFILE, SHOW CREATE PROFILE

Привилегия ROLE ADMIN разрешает пользователю назначать и отзывать любые роли, включая те, которые не назначены пользователю с опцией администратора.

SYSTEM

Разрешает выполнять запросы [SYSTEM](https://clickhouse.tech/docs/ru/sql-reference/statements/system/) в соответствии со следующей иерархией привилегий:

* SYSTEM. Уровень: GROUP:
  + SYSTEM SHUTDOWN. Уровень: GLOBAL. Алиасы: SYSTEM KILL, SHUTDOWN
  + SYSTEM DROP CACHE. Алиасы: DROP CACHE:
    - SYSTEM DROP DNS CACHE. Уровень: GLOBAL. Алиасы: SYSTEM DROP DNS, DROP DNS CACHE, DROP DNS
    - SYSTEM DROP MARK CACHE. Уровень: GLOBAL. Алиасы: SYSTEM DROP MARK, DROP MARK CACHE, DROP MARKS
    - SYSTEM DROP UNCOMPRESSED CACHE. Уровень: GLOBAL. Алиасы: SYSTEM DROP UNCOMPRESSED, DROP UNCOMPRESSED CACHE, DROP UNCOMPRESSED
  + SYSTEM RELOAD. Уровень: GROUP:
    - SYSTEM RELOAD CONFIG. Уровень: GLOBAL. Алиасы: RELOAD CONFIG
    - SYSTEM RELOAD DICTIONARY. Уровень: GLOBAL. Алиасы: SYSTEM RELOAD DICTIONARIES, RELOAD DICTIONARY, RELOAD DICTIONARIES
    - SYSTEM RELOAD EMBEDDED DICTIONARIES. Уровень: GLOBAL. Алиасы: RELOAD EMBEDDED DICTIONARIES
  + SYSTEM MERGES. Уровень: TABLE. Алиасы: SYSTEM STOP MERGES, SYSTEM START MERGES, STOP MERGES, START MERGES
  + SYSTEM TTL MERGES. Уровень: TABLE. Алиасы: SYSTEM STOP TTL MERGES, SYSTEM START TTL MERGES, STOP TTL MERGES, START TTL MERGES
  + SYSTEM FETCHES. Уровень: TABLE. Алиасы: SYSTEM STOP FETCHES, SYSTEM START FETCHES, STOP FETCHES, START FETCHES
  + SYSTEM MOVES. Уровень: TABLE. Алиасы: SYSTEM STOP MOVES, SYSTEM START MOVES, STOP MOVES, START MOVES
  + SYSTEM SENDS. Уровень: GROUP. Алиасы: SYSTEM STOP SENDS, SYSTEM START SENDS, STOP SENDS, START SENDS:
    - SYSTEM DISTRIBUTED SENDS. Уровень: TABLE. Алиасы: SYSTEM STOP DISTRIBUTED SENDS, SYSTEM START DISTRIBUTED SENDS, STOP DISTRIBUTED SENDS, START DISTRIBUTED SENDS
    - SYSTEM REPLICATED SENDS. Уровень: TABLE. Алиасы: SYSTEM STOP REPLICATED SENDS, SYSTEM START REPLICATED SENDS, STOP REPLICATED SENDS, START REPLICATED SENDS
  + SYSTEM REPLICATION QUEUES. Уровень: TABLE. Алиасы: SYSTEM STOP REPLICATION QUEUES, SYSTEM START REPLICATION QUEUES, STOP REPLICATION QUEUES, START REPLICATION QUEUES
  + SYSTEM SYNC REPLICA. Уровень: TABLE. Алиасы: SYNC REPLICA
  + SYSTEM RESTART REPLICA. Уровень: TABLE. Алиасы: RESTART REPLICA
  + SYSTEM FLUSH. Уровень: GROUP:
    - SYSTEM FLUSH DISTRIBUTED. Уровень: TABLE. Алиасы: FLUSH DISTRIBUTED
    - SYSTEM FLUSH LOGS. Уровень: GLOBAL. Алиасы: FLUSH LOGS

Привилегия SYSTEM RELOAD EMBEDDED DICTIONARIES имплицитно присваивается привилегией SYSTEM RELOAD DICTIONARY ON \*.\*.

INTROSPECTION

Разрешает использовать функции [интроспекции](https://clickhouse.tech/docs/ru/operations/optimizing-performance/sampling-query-profiler/):

* INTROSPECTION. Уровень: GROUP. Алиасы: INTROSPECTION FUNCTIONS:
  + addressToLine. Уровень: GLOBAL
  + addressToSymbol. Уровень: GLOBAL
  + demangle. Уровень: GLOBAL

SOURCES

Разрешает использовать внешние источники данных. Применяется к [движкам таблиц](https://clickhouse.tech/docs/ru/engines/table-engines/) и [табличным функциям](https://clickhouse.tech/docs/ru/sql-reference/table-functions/#table-functions):

* SOURCES. Уровень: GROUP:
  + FILE. Уровень: GLOBAL
  + URL. Уровень: GLOBAL
  + REMOTE. Уровень: GLOBAL
  + YSQL. Уровень: GLOBAL
  + ODBC. Уровень: GLOBAL
  + JDBC. Уровень: GLOBAL
  + HDFS. Уровень: GLOBAL
  + S3. Уровень: GLOBAL

Привилегия SOURCES разрешает использование всех источников. Также вы можете присвоить привилегию для каждого источника отдельно. Для использования источников необходимы дополнительные привилегии.

Примеры:

* Чтобы создать таблицу с [движком MySQL](https://clickhouse.tech/docs/ru/engines/table-engines/integrations/mysql/), необходимы привилегии CREATE TABLE (ON db.table\_name) и MYSQL.
* Чтобы использовать [табличную функцию mysql](https://clickhouse.tech/docs/ru/sql-reference/table-functions/mysql/), необходимы привилегии CREATE TEMPORARY TABLE и MYSQL.

dictGet

* dictGet. Алиасы: dictHas, dictGetHierarchy, dictIsIn.

Разрешает вызывать функции [dictGet](https://clickhouse.tech/docs/ru/sql-reference/functions/ext-dict-functions/#dictget), [dictHas](https://clickhouse.tech/docs/ru/sql-reference/functions/ext-dict-functions/#dicthas), [dictGetHierarchy](https://clickhouse.tech/docs/ru/sql-reference/functions/ext-dict-functions/#dictgethierarchy), [dictIsIn](https://clickhouse.tech/docs/ru/sql-reference/functions/ext-dict-functions/#dictisin).

Уровень: DICTIONARY.

Примеры:

* GRANT dictGet ON mydb.mydictionary TO john
* GRANT dictGet ON mydictionary TO john

ALL

Присваивает пользователю или роли все привилегии на объект с регулируемым доступом.

NONE

Не присваивает никаких привилегий.

ADMIN OPTION

Привилегия ADMIN OPTION разрешает пользователю назначать свои роли другому пользователю.

##### REVOKE

Отзывает привилегии у пользователей или ролей.

Синтаксис:

Отзыв привилегий у пользователей:

**REVOKE** [**ON** **CLUSTER** cluster\_name] privilege[(**column\_name** [,...])] [,...] **ON** {db.**table**|db.\*|\*.\*|**table**|\*} **FROM** {**user** | **CURRENT\_USER**} [,...] | **ALL** | **ALL** **EXCEPT** {**user** | **CURRENT\_USER**} [,...]

Отзыв ролей у пользователей:

**REVOKE** [**ON** **CLUSTER** cluster\_name] [**ADMIN** **OPTION** **FOR**] **role** [,...] **FROM** {**user** | **role** | **CURRENT\_USER**} [,...] | **ALL** | **ALL** **EXCEPT** {user\_name | role\_name | **CURRENT\_USER**} [,...]

Описание:

Для отзыва привилегий можно использовать привилегию более широкой области действия. Например, если у пользователя есть привилегия SELECT (x,y), администратор может отозвать ее с помощью одного из запросов: REVOKE SELECT(x,y) ..., REVOKE SELECT \* ... или даже REVOKE ALL PRIVILEGES ....

Частичный отзыв

Вы можете отозвать часть привилегии. Например, если у пользователя есть привилегия SELECT \*.\*, вы можете отозвать привилегию на чтение данных из какой-то таблицы или базы данных.

Примеры:

Присвоить пользователю john привилегию на SELECT из всех баз данных кроме accounts:

**GRANT** **SELECT** **ON** \*.\* **TO** john;

**REVOKE** **SELECT** **ON** accounts.\* **FROM** john;

Присвоить пользователю mira привилегию на SELECT из всех столбцов таблицы accounts.staff кроме столбца wage:

**GRANT** **SELECT** **ON** accounts.staff **TO** mira;

**REVOKE** **SELECT**(wage) **ON** accounts.staff **FROM** mira;

##### ATTACH Statement

Запрос полностью аналогичен запросу CREATE, но:

* вместо слова CREATE используется слово ATTACH;
* запрос не создаёт данные на диске, а предполагает, что данные уже лежат в соответствующих местах, и всего лишь добавляет информацию о таблице на сервер. После выполнения запроса ATTACH сервер будет знать о существовании таблицы.

Если таблица перед этим была отсоединена (DETACH), т.е. её структура известна, можно использовать сокращенную форму записи без определения структуры.

ATTACH **TABLE** [**IF** **NOT** **EXISTS**] [db.]name [**ON** **CLUSTER** **cluster**]

Этот запрос используется при старте сервера. Сервер хранит метаданные таблиц в виде файлов с запросами ATTACH, которые он просто исполняет при запуске (за исключением системных таблиц, которые явно создаются на сервере).

##### CHECK TABLE Statement

Проверяет таблицу на повреждение данных.

Синтаксис:

**CHECK** **TABLE** [db.]name

Запрос CHECK TABLE сравнивает текущие размеры файлов (в которых хранятся данные из колонок) с ожидаемыми значениями. Если значения не совпадают, данные в таблице считаются поврежденными. Искажение возможно, например, из-за сбоя при записи данных.

Ответ содержит колонку result, содержащую одну строку с типом [Boolean](https://clickhouse.tech/docs/ru/sql-reference/data-types/boolean/). Допустимые значения:

* 0 – данные в таблице повреждены,
* 1 – данные не повреждены.

Запрос CHECK TABLE поддерживает следующие движки таблиц:

* [Log](https://clickhouse.tech/docs/ru/engines/table-engines/log-family/log/),
* [TinyLog](https://clickhouse.tech/docs/ru/engines/table-engines/log-family/tinylog/),
* [StripeLog](https://clickhouse.tech/docs/ru/engines/table-engines/log-family/stripelog/),
* [Семейство MergeTree](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/).

При попытке выполнить запрос с таблицами с другими табличными движками, RT.WideStore генерирует исключение.

В движках \*Log не предусмотрено автоматическое восстановление данных после сбоя. Используйте запрос CHECK TABLE, чтобы своевременно выявлять повреждение данных.

Для движков из семейства MergeTree запрос CHECK TABLE показывает статус проверки для каждого отдельного куска данных таблицы на локальном сервере.

Что делать, если данные повреждены

В этом случае можно скопировать оставшиеся неповрежденные данные в другую таблицу. Для этого:

1. Создайте новую таблицу с такой же структурой, как у поврежденной таблицы. Для этого выполните запрос CREATE TABLE <new\_table\_name> AS <damaged\_table\_name>.
2. Установите значение параметра [max\_threads](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-max_threads) в 1. Это нужно для того, чтобы выполнить следующий запрос в одном потоке. Установить значение параметра можно через запрос: SET max\_threads = 1.
3. Выполните запрос INSERT INTO <new\_table\_name> SELECT \* FROM <damaged\_table\_name>. В результате неповрежденные данные будут скопированы в другую таблицу. Обратите внимание, будут скопированы только те данные, которые следуют до поврежденного участка.
4. Перезапустите clickhouse-client, чтобы вернуть предыдущее значение параметра max\_threads.

##### DESCRIBE TABLE Statement

Синтаксис:

**DESC**|**DESCRIBE** **TABLE** [db.]**table** [**INTO** OUTFILE filename] [FORMAT format]

Возвращает описание столбцов таблицы.

Результат запроса содержит столбцы (все столбцы имеют тип String):

* name – имя столбца таблицы;
* type – тип столбца;
* default\_type – в каком виде задано [выражение для значения по умолчанию](https://clickhouse.tech/docs/ru/sql-reference/statements/create/table/#create-default-values): DEFAULT, MATERIALIZED или ALIAS. Столбец содержит пустую строку, если значение по умолчанию не задано.
* default\_expression – значение, заданное в секции DEFAULT;
* comment\_expression – комментарий к столбцу.

Вложенные структуры данных выводятся в «развёрнутом» виде. То есть, каждый столбец - по отдельности, с именем через точку.

##### DETACH

Удаляет из сервера информацию о таблице name. Сервер перестаёт знать о существовании таблицы.

Синтаксис:

DETACH **TABLE** [**IF** **EXISTS**] [db.]name [**ON** **CLUSTER** **cluster**]

Но ни данные, ни метаданные таблицы не удаляются. При следующем запуске сервера, сервер прочитает метаданные и снова узнает о таблице.  
Также, «отцепленную» таблицу можно прицепить заново запросом ATTACH (за исключением системных таблиц, для которых метаданные не хранятся).

Запроса DETACH DATABASE нет.

##### DROP

Удаляет существующий объект. Если указано IF EXISTS - не выдавать ошибку, если объекта не существует.

###### DROP DATABASE

Удаляет все таблицы в базе данных db, затем удаляет саму базу данных db.

Синтаксис:

**DROP** **DATABASE** [**IF** **EXISTS**] db [**ON** **CLUSTER** **cluster**]

###### DROP TABLE

Удаляет таблицу.

Синтаксис:

**DROP** [**TEMPORARY**] **TABLE** [**IF** **EXISTS**] [db.]name [**ON** **CLUSTER** **cluster**]

###### DROP DICTIONARY

Удаляет словарь.

Синтаксис:

**DROP** **DICTIONARY** [**IF** **EXISTS**] [db.]name

###### DROP USER

Удаляет пользователя.

Синтаксис:

**DROP** **USER** [**IF** **EXISTS**] name [,...] [**ON** **CLUSTER** cluster\_name]

###### DROP ROLE

Удаляет роль. При удалении роль отзывается у всех объектов системы доступа, которым она присвоена.

Синтаксис:

**DROP** **ROLE** [**IF** **EXISTS**] name [,...] [**ON** **CLUSTER** cluster\_name]

###### DROP ROW POLICY

Удаляет политику доступа к строкам. При удалении политика отзывается у всех объектов системы доступа, которым она присвоена.

Синтаксис:

**DROP** [**ROW**] POLICY [**IF** **EXISTS**] name [,...] **ON** [**database**.]**table** [,...] [**ON** **CLUSTER** cluster\_name]

###### DROP QUOTA

Удаляет квоту. При удалении квота отзывается у всех объектов системы доступа, которым она присвоена.

Синтаксис:

**DROP** QUOTA [**IF** **EXISTS**] name [,...] [**ON** **CLUSTER** cluster\_name]

###### DROP SETTINGS PROFILE

Удаляет профиль настроек. При удалении профиль отзывается у всех объектов системы доступа, которым он присвоен.

Синтаксис:

**DROP** [SETTINGS] PROFILE [**IF** **EXISTS**] name [,...] [**ON** **CLUSTER** cluster\_name]

###### DROP VIEW

Удаляет представление. Представления могут быть удалены и командой DROP TABLE, но команда DROP VIEW проверяет, что [db.]name является представлением.

Синтаксис:

**DROP** **VIEW** [**IF** **EXISTS**] [db.]name [**ON** **CLUSTER** **cluster**]

##### EXISTS

**EXISTS** [**TEMPORARY**] **TABLE** [db.]name [**INTO** OUTFILE filename] [FORMAT format]

Возвращает один столбец типа UInt8, содержащий одно значение - 0, если таблицы или БД не существует и 1, если таблица в указанной БД существует.

##### KILL

Существует два вида операторов KILL: KILL QUERY и KILL MUTATION.

###### KILL QUERY

KILL QUERY [**ON** **CLUSTER** **cluster**]

**WHERE** <**where** expression **to** **SELECT** **FROM** **system**.processes query>

[SYNC|ASYNC|TEST]

[FORMAT format]

Пытается принудительно остановить исполняющиеся в данный момент запросы.

Запросы для принудительной остановки выбираются из таблицы system.processes с помощью условия, указанного в секции WHERE запроса KILL.

Примеры:

-- Принудительно останавливает все запросы с указанным query\_id:

KILL QUERY **WHERE** query\_id='2-857d-4a57-9ee0-327da5d60a90'

-- Синхронно останавливает все запросы пользователя 'username':

KILL QUERY **WHERE** **user**='username' SYNC

Readonly-пользователи могут останавливать только свои запросы.

По умолчанию используется асинхронный вариант запроса (ASYNC), который не дожидается подтверждения остановки запросов.

Синхронный вариант (SYNC) ожидает остановки всех запросов и построчно выводит информацию о процессах по ходу их остановки.

Ответ содержит колонку kill\_status, которая может принимать следующие значения:

1. ‘finished’ – запрос был успешно остановлен;
2. ‘waiting’ – запросу отправлен сигнал завершения, ожидается его остановка;
3. остальные значения описывают причину невозможности остановки запроса.

Тестовый вариант запроса (TEST) только проверяет права пользователя и выводит список запросов для остановки.

###### KILL MUTATION

KILL MUTATION [**ON** **CLUSTER** **cluster**]

**WHERE** <**where** expression **to** **SELECT** **FROM** **system**.mutations query>

[TEST]

[FORMAT format]

Пытается остановить выполняющиеся в данные момент [мутации](https://clickhouse.tech/docs/ru/sql-reference/statements/alter/#mutations). Мутации для остановки выбираются из таблицы [system.mutations](https://clickhouse.tech/docs/ru/operations/system-tables/mutations/#system_tables-mutations) с помощью условия, указанного в секции WHERE запроса KILL.

Тестовый вариант запроса (TEST) только проверяет права пользователя и выводит список запросов для остановки.

Примеры:

-- Останавливает все мутации одной таблицы:

KILL MUTATION **WHERE** **database** = 'default' **AND** **table** = 'table'

-- Останавливает конкретную мутацию:

KILL MUTATION **WHERE** **database** = 'default' **AND** **table** = 'table' **AND** mutation\_id = 'mutation\_3.txt'

Запрос полезен в случаях, когда мутация не может выполниться до конца (например, если функция в запросе мутации бросает исключение на данных таблицы).

Данные, уже изменённые мутацией, остаются в таблице (отката на старую версию данных не происходит).

##### OPTIMIZE

OPTIMIZE **TABLE** [db.]name [**ON** **CLUSTER** **cluster**] [PARTITION partition | PARTITION ID 'partition\_id'] [**FINAL**] [DEDUPLICATE]

Запрос пытается запустить внеплановый мёрж кусков данных для таблиц семейства [MergeTree](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/mergetree/). Другие движки таблиц не поддерживаются.

Если OPTIMIZE применяется к таблицам семейства [ReplicatedMergeTree](https://clickhouse.tech/docs/ru/engines/table-engines/mergetree-family/replication/), RT.WideStore создаёт задачу на мёрж и ожидает её исполнения на всех узлах (если активирована настройка replication\_alter\_partitions\_sync).

* Если OPTIMIZE не выполняет мёрж по любой причине, RT.WideStore не оповещает об этом клиента. Чтобы включить оповещения, используйте настройку [optimize\_throw\_if\_noop](https://clickhouse.tech/docs/ru/operations/settings/settings/#setting-optimize_throw_if_noop).
* Если указать PARTITION, то оптимизация выполняется только для указанной партиции. [Как задавать имя партиции в запросах](https://clickhouse.tech/docs/ru/sql-reference/statements/alter/#alter-how-to-specify-part-expr).
* Если указать FINAL, то оптимизация выполняется даже в том случае, если все данные уже лежат в одном куске. Кроме того, слияние является принудительным, даже если выполняются параллельные слияния.
* Если указать DEDUPLICATE, то произойдет схлопывание полностью одинаковых строк (сравниваются значения во всех колонках), имеет смысл только для движка MergeTree.

***Внимание:*** *Запрос OPTIMIZE не может устранить причину появления ошибки «Too many parts».*

##### RENAME

Переименовывает одну или несколько таблиц.

**RENAME** **TABLE** [db11.]name11 **TO** [db12.]name12, [db21.]name21 **TO** [db22.]name22, ... [**ON** **CLUSTER** **cluster**]

Переименовывание таблицы является лёгкой операцией. Если вы указали после TO другую базу данных, то таблица будет перенесена в эту базу данных. При этом, директории с базами данных должны быть расположены в одной файловой системе (иначе возвращается ошибка). В случае переименования нескольких таблиц в одном запросе – это неатомарная операция, может выполнится частично, запросы в других сессиях могут получить ошибку Table ... doesn't exist....

##### SET

**SET** param = value

Устанавливает значение value для [настройки](https://clickhouse.tech/docs/ru/operations/settings/) param в текущей сессии. [Конфигурационные параметры сервера](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/) нельзя изменить подобным образом.

Можно одним запросом установить все настройки из заданного профиля настроек.

**SET** profile = 'profile-name-from-the-settings-file'

Подробности смотрите в разделе [Настройки](https://clickhouse.tech/docs/ru/operations/settings/settings/).

##### SET ROLE

Активирует роли для текущего пользователя.

Синтаксис:

**SET** **ROLE** {**DEFAULT** | **NONE** | **role** [,...] | **ALL** | **ALL** **EXCEPT** **role** [,...]}

###### SET DEFAULT ROLE

Устанавливает роли по умолчанию для пользователя.

Роли по умолчанию активируются автоматически при входе пользователя. Ролями по умолчанию могут быть установлены только ранее назначенные роли. Если роль не назначена пользователю, RT.WideStore выбрасывает исключение.

Синтаксис:

**SET** **DEFAULT** **ROLE** {**NONE** | **role** [,...] | **ALL** | **ALL** **EXCEPT** **role** [,...]} **TO** {**user**|**CURRENT\_USER**} [,...]

Примеры[:](https://clickhouse.tech/docs/ru/sql-reference/statements/set-role/#set-default-role-examples)

Установить несколько ролей по умолчанию для пользователя:

**SET** **DEFAULT** **ROLE** role1, role2, ... **TO** **user**

Установить ролями по умолчанию все назначенные пользователю роли:

**SET** **DEFAULT** **ROLE** **ALL** **TO** **user**

Удалить роли по умолчанию для пользователя:

**SET** **DEFAULT** **ROLE** **NONE** **TO** **user**

Установить ролями по умолчанию все назначенные пользователю роли за исключением указанных:

**SET** **DEFAULT** **ROLE** **ALL** **EXCEPT** role1, role2 **TO** **user**

##### TRUNCATE

**TRUNCATE** **TABLE** [**IF** **EXISTS**] [db.]name [**ON** **CLUSTER** **cluster**]

Удаляет все данные из таблицы. Если условие IF EXISTS не указано, запрос вернет ошибку, если таблицы не существует.

Запрос TRUNCATE не поддерживается для следующих движков: [View](https://clickhouse.tech/docs/ru/engines/table-engines/special/view/), [File](https://clickhouse.tech/docs/ru/engines/table-engines/special/file/), [URL](https://clickhouse.tech/docs/ru/engines/table-engines/special/url/) и [Null](https://clickhouse.tech/docs/ru/engines/table-engines/special/null/).

##### USE

USE db

Позволяет установить текущую базу данных для сессии.

Текущая база данных используется для поиска таблиц, если база данных не указана в запросе явно через точку перед именем таблицы.

При использовании HTTP протокола запрос не может быть выполнен, так как понятия сессии не существует.

ПРИЛОЖЕНИЕ 2. ОПИСАНИЕ ФУНКЦИЙ В RT.WIDESTORE

##### Арифметические функции

Для всех арифметических функций, тип результата вычисляется, как минимальный числовой тип, который может вместить результат, если такой тип есть. Минимум берётся одновременно по числу бит, знаковости и «плавучести». Если бит не хватает, то берётся тип максимальной битности.

Пример:

**SELECT** toTypeName(0), toTypeName(0 + 0), toTypeName(0 + 0 + 0), toTypeName(0 + 0 + 0 + 0)

┌─toTypeName(0)─┬─toTypeName(plus(0, 0))─┬─toTypeName(plus(plus(0, 0), 0))─┬─toTypeName(plus(plus(plus(0, 0), 0), 0))─┐

│ UInt8 │ UInt16 │ UInt32 │ UInt64 │

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

Арифметические функции работают для любой пары типов из UInt8, UInt16, UInt32, UInt64, Int8, Int16, Int32, Int64, Float32, Float64.

Переполнение производится также, как в C++.

###### plus(a, b), оператор a + b

Вычисляет сумму чисел.

Также можно складывать целые числа с датой и датой-с-временем. В случае даты, прибавление целого числа означает прибавление соответствующего количества дней. В случае даты-с-временем - прибавление соответствующего количества секунд.

###### minus(a, b), оператор a - b

Вычисляет разность чисел. Результат всегда имеет знаковый тип.

Также можно вычитать целые числа из даты и даты-с-временем. Смысл аналогичен - смотрите выше для plus.

###### multiply(a, b), оператор a \* b

Вычисляет произведение чисел.

###### divide(a, b), оператор a / b

Вычисляет частное чисел. Тип результата всегда является типом с плавающей запятой.

То есть, деление не целочисленное. Для целочисленного деления, используйте функцию intDiv.

При делении на ноль получится inf, -inf или nan.

###### intDiv(a, b)

Вычисляет частное чисел. Деление целочисленное, с округлением вниз (по абсолютному значению).

При делении на ноль или при делении минимального отрицательного числа на минус единицу, кидается исключение.

###### intDivOrZero(a, b)

Отличается от intDiv тем, что при делении на ноль или при делении минимального отрицательного числа на минус единицу, возвращается ноль.

###### modulo(a, b), оператор a % b

Вычисляет остаток от деления.

Если аргументы – числа с плавающей запятой, то они предварительно преобразуются в целые числа, путём отбрасывания дробной части.

Берётся остаток в том же смысле, как это делается в C++. По факту, для отрицательных чисел, используется truncated division.

При делении на ноль или при делении минимального отрицательного числа на минус единицу, кидается исключение.

###### moduloOrZero(a, b)

В отличие от [modulo](https://clickhouse.tech/docs/ru/sql-reference/functions/arithmetic-functions/#modulo), возвращает ноль при делении на ноль.

###### negate(a), оператор -a

Вычисляет число, обратное по знаку. Результат всегда имеет знаковый тип.

###### abs(a)

Вычисляет абсолютное значение для числа a. То есть, если a \< 0, то возвращает -a.

Для беззнаковых типов ничего не делает. Для чисел типа целых со знаком, возвращает число беззнакового типа.

###### gcd(a, b)

Вычисляет наибольший общий делитель чисел.

При делении на ноль или при делении минимального отрицательного числа на минус единицу, кидается исключение.

###### lcm(a, b)

Вычисляет наименьшее общее кратное чисел.

При делении на ноль или при делении минимального отрицательного числа на минус единицу, кидается исключение.

##### Массивы

###### empty

Возвращает 1 для пустого массива, и 0 для непустого массива.

Тип результата – UInt8.

Функция также работает для строк.

###### notEmpty

Возвращает 0 для пустого массива, и 1 для непустого массива.

Тип результата – UInt8.

Функция также работает для строк.

###### length

Возвращает количество элементов в массиве.

Тип результата – UInt64.

Функция также работает для строк.

###### emptyArrayUInt8, emptyArrayUInt16, emptyArrayUInt32, emptyArrayUInt64

###### emptyArrayInt8, emptyArrayInt16, emptyArrayInt32, emptyArrayInt64

###### emptyArrayFloat32, emptyArrayFloat64

###### emptyArrayDate, emptyArrayDateTime

###### emptyArrayString

Принимает ноль аргументов и возвращает пустой массив соответствующего типа.

###### emptyArrayToSingle

Принимает пустой массив и возвращает массив из одного элемента, равного значению по умолчанию.

###### range(N)

Возвращает массив чисел от 0 до N-1.

На всякий случай, если на блок данных, создаются массивы суммарной длины больше 100 000 000 элементов, то кидается исключение.

###### array(x1, …), оператор [x1, …]

Создаёт массив из аргументов функции.

Аргументы должны быть константами и иметь типы, для которых есть наименьший общий тип. Должен быть передан хотя бы один аргумент, так как иначе непонятно, какого типа создавать массив. То есть, с помощью этой функции невозможно создать пустой массив (для этого используйте функции emptyArray\*, описанные выше).

Возвращает результат типа Array(T), где T – наименьший общий тип от переданных аргументов.

###### arrayConcat

Объединяет массивы, переданные в качестве аргументов.

arrayConcat(arrays)

Параметры:

* arrays – произвольное количество элементов типа [Array](https://clickhouse.tech/docs/ru/sql-reference/functions/array-functions/).

Пример:

**SELECT** arrayConcat([1, 2], [3, 4], [5, 6]) **AS** res

┌─res───────────┐

│ [1,2,3,4,5,6] │

└───────────────┘

###### arrayElement(arr, n), operator arr[n]

Достаёт элемент с индексом n из массива arr. n должен быть любым целочисленным типом.

Индексы в массиве начинаются с единицы.

Поддерживаются отрицательные индексы. В этом случае, будет выбран соответствующий по номеру элемент с конца. Например, arr[-1] - последний элемент массива.

Если индекс выходит за границы массива, то возвращается некоторое значение по умолчанию (0 для чисел, пустая строка для строк и т. п.), кроме случая с неконстантным массивом и константным индексом 0 (в этом случае будет ошибка Array indices are 1-based).

###### has(arr, elem)

Проверяет наличие элемента elem в массиве arr.

Возвращает 0, если элемента в массиве нет, или 1, если есть.

NULL обрабатывается как значение.

**SELECT** has([1, 2, **NULL**], **NULL**)

┌─has([1, 2, NULL], NULL)─┐

│ 1 │

└─────────────────────────┘

###### hasAll

Проверяет, является ли один массив подмножеством другого.

Синтаксис:

hasAll(**set**, subset)

Параметры:

* set – массив любого типа с набором элементов.
* subset – массив любого типа со значениями, которые проверяются на вхождение в set.

Возвращаемые значения:

* 1, если set содержит все элементы из subset.
* 0, в противном случае.

Особенности:

* Пустой массив является подмножеством любого массива.
* NULL обрабатывается как значение.
* Порядок значений в обоих массивах не имеет значения.

Примеры:

SELECT hasAll([], []) возвращает 1.

SELECT hasAll([1, Null], [Null]) возвращает 1.

SELECT hasAll([1.0, 2, 3, 4], [1, 3]) возвращает 1.

SELECT hasAll(['a', 'b'], ['a']) возвращает 1.

SELECT hasAll([1], ['a']) возвращает 0.

SELECT hasAll([[1, 2], [3, 4]], [[1, 2], [3, 5]]) возвращает 0.

###### hasAny

Проверяет, имеют ли два массива хотя бы один общий элемент.

Синтаксис:

hasAny(array1, array2)

Параметры:

array1 – массив любого типа с набором элементов.

array2 – массив любого типа с набором элементов.

Возвращаемые значения:

* 1, если array1 и array2 имеют хотя бы один одинаковый элемент.
* 0, в противном случае.

Особенности:

* NULL обрабатывается как значение.
* Порядок значений в обоих массивах не имеет значения.

Примеры:

SELECT hasAny([1], []) возвращает 0.

SELECT hasAny([Null], [Null, 1]) возвращает 1.

SELECT hasAny([-128, 1., 512], [1]) возвращает 1.

SELECT hasAny([[1, 2], [3, 4]], ['a', 'c']) возвращает 0.

SELECT hasAll([[1, 2], [3, 4]], [[1, 2], [1, 2]]) возвращает 1.

###### indexOf(arr, x)

Возвращает индекс первого элемента x (начиная с 1), если он есть в массиве, или 0, если его нет.

Пример:

**SELECT** indexOf([1, 3, **NULL**, **NULL**], **NULL**)

┌─indexOf([1, 3, NULL, NULL], NULL)─┐

│ 3 │

└───────────────────────────────────┘

Элементы, равные NULL, обрабатываются как обычные значения.

###### arrayCount([func,] arr1, …)

Возвращает количество элементов массива arr, для которых функция func возвращает не 0. Если func не указана - возвращает количество ненулевых элементов массива.

Функция arrayCount является [функцией высшего порядка](https://clickhouse.tech/docs/ru/sql-reference/functions/#higher-order-functions) – в качестве первого аргумента ей можно передать лямбда-функцию.

###### countEqual(arr, x)

Возвращает количество элементов массива, равных x. Эквивалентно arrayCount(elem -> elem = x, arr).

NULL обрабатывается как значение.

Пример:

**SELECT** countEqual([1, 2, **NULL**, **NULL**], **NULL**)

┌─countEqual([1, 2, NULL, NULL], NULL)─┐

│ 2 │

└──────────────────────────────────────┘

###### arrayEnumerate(arr)

Возвращает массив [1, 2, 3, …, length(arr)].

Эта функция обычно используется совместно с ARRAY JOIN. Она позволяет, после применения ARRAY JOIN, посчитать что-либо только один раз для каждого массива. Пример:

**SELECT**

**count**() **AS** Reaches,

countIf(num = 1) **AS** Hits

**FROM** test.hits

ARRAY **JOIN**

GoalsReached,

arrayEnumerate(GoalsReached) **AS** num

**WHERE** CounterID = 160656

**LIMIT** 10

┌─Reaches─┬──Hits─┐

│ 95606 │ 31406 │

└─────────┴───────┘

В этом примере, Reaches – число достижений целей (строк, получившихся после применения ARRAY JOIN), а Hits – число хитов (строк, которые были до ARRAY JOIN). В данном случае, тот же результат можно получить проще:

**SELECT**

**sum**(**length**(GoalsReached)) **AS** Reaches,

**count**() **AS** Hits

**FROM** test.hits

**WHERE** (CounterID = 160656) **AND** notEmpty(GoalsReached)

┌─Reaches─┬──Hits─┐

│ 95606 │ 31406 │

└─────────┴───────┘

Также эта функция может быть использована в функциях высшего порядка. Например, с её помощью можно достать индексы массива для элементов, удовлетворяющих некоторому условию.

###### arrayEnumerateUniq(arr, …)

Возвращает массив, такого же размера, как исходный, где для каждого элемента указано, какой он по счету среди элементов с таким же значением.

Например: arrayEnumerateUniq([10, 20, 10, 30]) = [1, 1, 2, 1].

Эта функция полезна при использовании ARRAY JOIN и агрегации по элементам массива.

Пример:

**SELECT**

Goals.ID **AS** GoalID,

**sum**(Sign) **AS** Reaches,

sumIf(Sign, num = 1) **AS** Visits

**FROM** test.visits

ARRAY **JOIN**

Goals,

arrayEnumerateUniq(Goals.ID) **AS** num

**WHERE** CounterID = 160656

**GROUP** **BY** GoalID

**ORDER** **BY** Reaches **DESC**

**LIMIT** 10

┌──GoalID─┬─Reaches─┬─Visits─┐

│ 53225 │ 3214 │ 1097 │

│ 2825062 │ 3188 │ 1097 │

│ 56600 │ 2803 │ 488 │

│ 1989037 │ 2401 │ 365 │

│ 2830064 │ 2396 │ 910 │

│ 1113562 │ 2372 │ 373 │

│ 3270895 │ 2262 │ 812 │

│ 1084657 │ 2262 │ 345 │

│ 56599 │ 2260 │ 799 │

│ 3271094 │ 2256 │ 812 │

└─────────┴─────────┴────────┘

В этом примере, для каждого идентификатора цели, посчитано количество достижений целей (каждый элемент вложенной структуры данных Goals является достижением целей) и количество визитов. Если бы не было ARRAY JOIN, мы бы считали количество визитов как sum(Sign). Но в данном случае, строчки были размножены по вложенной структуре Goals, и чтобы после этого учесть каждый визит один раз, мы поставили условие на значение функции arrayEnumerateUniq(Goals.ID).

Функция arrayEnumerateUniq может принимать несколько аргументов - массивов одинаковых размеров. В этом случае, уникальность считается для кортежей элементов на одинаковых позициях всех массивов:

**SELECT** arrayEnumerateUniq([1, 1, 1, 2, 2, 2], [1, 1, 2, 1, 1, 2]) **AS** res

┌─res───────────┐

│ [1,2,1,1,2,1] │

└───────────────┘

Это нужно при использовании ARRAY JOIN с вложенной структурой данных и затем агрегации по нескольким элементам этой структуры.

###### arrayPopBack

Удаляет последний элемент из массива.

Синтаксис:

arrayPopBack(array)

Параметры:

* array – массив.

Пример:

**SELECT** arrayPopBack([1, 2, 3]) **AS** res

text

┌─res───┐

│ [1,2] │

└───────┘

###### arrayPopFront

Удаляет первый элемент из массива:

arrayPopFront(array)

Параметры:

* array – массив.

Пример:

**SELECT** arrayPopFront([1, 2, 3]) **AS** res

┌─res───┐

│ [2,3] │

└───────┘

###### arrayPushBack

Добавляет один элемент в конец массива.

Синтаксис:

arrayPushBack(array, single\_value)

Параметры:

* array – массив.
* single\_value – одиночное значение. В массив с числами можно добавить только числа, в массив со строками только строки. При добавлении чисел RT.WideStore автоматически приводит тип single\_value к типу данных массива. Подробнее о типах данных в RT.WideStore читайте в разделе «[Типы данных](https://clickhouse.tech/docs/ru/sql-reference/functions/array-functions/#data_types)». Может быть равно NULL. Функция добавит элемент NULL в массив, а тип элементов массива преобразует в Nullable.

Пример:

**SELECT** arrayPushBack(['a'], 'b') **AS** res

┌─res───────┐

│ ['a','b'] │

└───────────┘

###### arrayPushFront

Добавляет один элемент в начало массива.

Синтаксис:

arrayPushFront(array, single\_value)

Параметры:

* array – массив.
* single\_value – одиночное значение. В массив с числами можно добавить только числа, в массив со строками только строки. При добавлении чисел RT.WideStore автоматически приводит тип single\_value к типу данных массива. Подробнее о типах данных в RT.WideStore читайте в разделе «[Типы данных](https://clickhouse.tech/docs/ru/sql-reference/functions/array-functions/#data_types)». Может быть равно NULL. Функция добавит элемент NULL в массив, а тип элементов массива преобразует в Nullable.

Пример:

**SELECT** arrayPushFront(['b'], 'a') **AS** res

┌─res───────┐

│ ['a','b'] │

└───────────┘

###### arrayResize

Изменяет длину массива.

Синтаксис:

arrayResize(array, **size**[, extender])

Параметры:

* array – массив.
* size – необходимая длина массива.
  + Если size меньше изначального размера массива, то массив обрезается справа.
  + Если size больше изначального размера массива, массив дополняется справа значениями extender или значениями по умолчанию для типа данных элементов массива.
* extender – значение для дополнения массива. Может быть NULL.

Возвращаемое значение:

Массив длины size.

Примеры вызовов:

**SELECT** arrayResize([1], 3)

┌─arrayResize([1], 3)─┐

│ [1,0,0] │

└─────────────────────┘

**SELECT** arrayResize([1], 3, **NULL**)

┌─arrayResize([1], 3, NULL)─┐

│ [1,NULL,NULL] │

└───────────────────────────┘

###### arraySlice

Возвращает срез массива.

Синтаксис:

arraySlice(array, **offset**[, **length**])

Параметры:

* array – массив данных.
* offset – отступ от края массива. Положительное значение - отступ слева, отрицательное значение - отступ справа. Отсчет элементов массива начинается с 1.
* length – длина необходимого среза. Если указать отрицательное значение, то функция вернёт открытый срез [offset, array\_length - length). Если не указать значение, то функция вернёт срез [offset, the\_end\_of\_array].

Пример:

**SELECT** arraySlice([1, 2, **NULL**, 4, 5], 2, 3) **AS** res

┌─res────────┐

│ [2,NULL,4] │

└────────────┘

Элементы массива равные NULL обрабатываются как обычные значения.

###### arraySort([func,] arr, …)

Возвращает массив arr, отсортированный в восходящем порядке. Если задана функция func, то порядок сортировки определяется результатом применения этой функции на элементы массива arr. Если func принимает несколько аргументов, то в функцию arraySort нужно передавать несколько массивов, которые будут соответствовать аргументам функции func. Подробные примеры рассмотрены в конце описания arraySort.

Пример сортировки целочисленных значений:

**SELECT** arraySort([1, 3, 3, 0])

┌─arraySort([1, 3, 3, 0])─┐

│ [0,1,3,3] │

└─────────────────────────┘

Пример сортировки строковых значений:

**SELECT** arraySort(['hello', 'world', '!'])

┌─arraySort(['hello', 'world', '!'])─┐

│ ['!','hello','world'] │

└────────────────────────────────────┘

Значения NULL, NaN и Inf сортируются по следующему принципу:

**SELECT** arraySort([1, nan, 2, **NULL**, 3, nan, -4, **NULL**, inf, -inf]);

┌─arraySort([1, nan, 2, NULL, 3, nan, -4, NULL, inf, -inf])─┐

│ [-inf,-4,1,2,3,inf,nan,nan,NULL,NULL] │

└───────────────────────────────────────────────────────────┘

* Значения -Inf идут в начале массива.
* Значения NULL идут в конце массива.
* Значения NaN идут перед NULL.
* Значения Inf идут перед NaN.

Функция arraySort является [функцией высшего порядка](https://clickhouse.tech/docs/ru/sql-reference/functions/#higher-order-functions) – в качестве первого аргумента ей можно передать лямбда-функцию. В этом случае порядок сортировки определяется результатом применения лямбда-функции на элементы массива.

Рассмотрим пример:

**SELECT** arraySort((x) -> -x, [1, 2, 3]) **as** res;

┌─res─────┐

│ [3,2,1] │

└─────────┘

Для каждого элемента исходного массива лямбда-функция возвращает ключ сортировки, то есть [1 –> -1, 2 –> -2, 3 –> -3]. Так как arraySort сортирует элементы в порядке возрастания ключей, результат будет [3, 2, 1]. Как можно заметить, функция x –> -x устанавливает [обратный порядок сортировки](https://clickhouse.tech/docs/ru/sql-reference/functions/array-functions/#array_functions-reverse-sort).

Лямбда-функция может принимать несколько аргументов. В этом случае, в функцию arraySort нужно передавать несколько массивов, которые будут соответствовать аргументам лямбда-функции (массивы должны быть одинаковой длины). Следует иметь в виду, что результат будет содержать элементы только из первого массива; элементы из всех последующих массивов будут задавать ключи сортировки. Например:

**SELECT** arraySort((x, y) -> y, ['hello', 'world'], [2, 1]) **as** res;

┌─res────────────────┐

│ ['world', 'hello'] │

└────────────────────┘

Элементы, указанные во втором массиве ([2,1]), определяют ключ сортировки для элементов из исходного массива ([‘hello’, ‘world’]), то есть [‘hello’ –> 2, ‘world’ –> 1]. Так как лямбда-функция не использует x, элементы исходного массива не влияют на порядок сортировки. Таким образом, ‘hello’ будет вторым элементом в отсортированном массиве, а ‘world’ – первым.

Ниже приведены другие примеры:

**SELECT** arraySort((x, y) -> y, [0, 1, 2], ['c', 'b', 'a']) **as** res;

┌─res─────┐

│ [2,1,0] │

└─────────┘

**SELECT** arraySort((x, y) -> -y, [0, 1, 2], [1, 2, 3]) **as** res;

┌─res─────┐

│ [2,1,0] │

└─────────┘

***Примечание:*** *Для улучшения эффективности сортировки применяется* [*преобразование Шварца*](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D0%BE%D0%B1%D1%80%D0%B0%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%A8%D0%B2%D0%B0%D1%80%D1%86%D0%B0)*.*

###### arrayReverseSort([func,] arr, …)

Возвращает массив arr, отсортированный в нисходящем порядке. Если указана функция func, то массив arr сначала сортируется в порядке, который определяется функцией func, а затем отсортированный массив переворачивается. Если функция func принимает несколько аргументов, то в функцию arrayReverseSort необходимо передавать несколько массивов, которые будут соответствовать аргументам функции func. Подробные примеры рассмотрены в конце описания функции arrayReverseSort.

Пример сортировки целочисленных значений:

**SELECT** arrayReverseSort([1, 3, 3, 0]);

┌─arrayReverseSort([1, 3, 3, 0])─┐

│ [3,3,1,0] │

└────────────────────────────────┘

Пример сортировки строковых значений:

**SELECT** arrayReverseSort(['hello', 'world', '!']);

┌─arrayReverseSort(['hello', 'world', '!'])─┐

│ ['world','hello','!'] │

└───────────────────────────────────────────┘

Значения NULL, NaN и Inf сортируются в следующем порядке:

**SELECT** arrayReverseSort([1, nan, 2, **NULL**, 3, nan, -4, **NULL**, inf, -inf]) **as** res;

┌─res───────────────────────────────────┐

│ [inf,3,2,1,-4,-inf,nan,nan,NULL,NULL] │

└───────────────────────────────────────┘

* Значения Inf идут в начале массива.
* Значения NULL идут в конце массива.
* Значения NaN идут перед NULL.
* Значения -Inf идут перед NaN.

Функция arrayReverseSort является [функцией высшего порядка](https://clickhouse.tech/docs/ru/sql-reference/functions/#higher-order-functions) – в качестве первого аргумента ей можно передать лямбда-функцию. Например:

**SELECT** arrayReverseSort((x) -> -x, [1, 2, 3]) **as** res;

┌─res─────┐

│ [1,2,3] │

└─────────┘

В этом примере, порядок сортировки устанавливается следующим образом:

1. Сначала исходный массив ([1, 2, 3]) сортируется в том порядке, который определяется лямбда-функцией. Результатом будет массив [3, 2, 1].
2. Массив, который был получен на предыдущем шаге, переворачивается. То есть, получается массив [1, 2, 3].

Лямбда-функция может принимать на вход несколько аргументов. В этом случае, в функцию arrayReverseSort нужно передавать несколько массивов, которые будут соответствовать аргументам лямбда-функции (массивы должны быть одинаковой длины). Следует иметь в виду, что результат будет содержать элементы только из первого массива; элементы из всех последующих массивов будут определять ключи сортировки. Например:

**SELECT** arrayReverseSort((x, y) -> y, ['hello', 'world'], [2, 1]) **as** res;

┌─res───────────────┐

│ ['hello','world'] │

└───────────────────┘

В этом примере, массив сортируется следующим образом:

1. Сначала массив сортируется в том порядке, который определяется лямбда-функцией. Элементы, указанные во втором массиве ([2,1]), определяют ключи сортировки соответствующих элементов из исходного массива ([‘hello’, ‘world’]). То есть, будет массив [‘world’, ‘hello’].
2. Массив, который был отсортирован на предыдущем шаге, переворачивается. Получается массив [‘hello’, ‘world’].

Ниже приведены ещё примеры:

**SELECT** arrayReverseSort((x, y) -> y, [0, 1, 2], ['c', 'b', 'a']) **as** res;

┌─res─────┐

│ [0,1,2] │

└─────────┘

**SELECT** arrayReverseSort((x, y) -> -y, [4, 3, 5], [1, 2, 3]) **AS** res;

┌─res─────┐

│ [4,3,5] │

└─────────┘

###### arrayUniq(arr, …)

Если передан один аргумент, считает количество разных элементов в массиве.  
Если передано несколько аргументов, считает количество разных кортежей из элементов на соответствующих позициях в нескольких массивах.

Если необходимо получить список уникальных элементов массива, можно воспользоваться arrayReduce(‘groupUniqArray’, arr).

###### arrayJoin(arr)

Особенная функция. Смотрите раздел [«Функция arrayJoin»](https://clickhouse.tech/docs/ru/sql-reference/functions/array-join/#functions_arrayjoin).

###### arrayDifference

Вычисляет разность между соседними элементами массива. Возвращает массив, где первым элементом будет 0, вторым – разность a[1] - a[0] и т. д. Тип элементов результирующего массива определяется правилами вывода типов при вычитании (напр. UInt8 - UInt8 = Int16).

Синтаксис:

arrayDifference(array)

Параметры:

* array – [массив](https://clickhouse.tech/docs/ru/data_types/array/).

Возвращаемое значение

Возвращает массив разностей между соседними элементами.

Пример:

Запрос:

**SELECT** arrayDifference([1, 2, 3, 4])

Ответ:

┌─arrayDifference([1, 2, 3, 4])─┐

│ [0,1,1,1] │

└───────────────────────────────┘

Пример переполнения из-за результирующего типа Int64:

Запрос:

**SELECT** arrayDifference([0, 10000000000000000000])

Ответ:

┌─arrayDifference([0, 10000000000000000000])─┐

│ [0,-8446744073709551616] │

└────────────────────────────────────────────┘

###### arrayDistinct

Принимает массив, возвращает массив, содержащий уникальные элементы.

Синтаксис:

arrayDistinct(array)

Параметры:

* array – [массив](https://clickhouse.tech/docs/ru/data_types/array/).

Возвращаемое значение:

Возвращает массив, содержащий только уникальные элементы исходного массива.

Пример:

Запрос:

**SELECT** arrayDistinct([1, 2, 2, 3, 1])

Ответ:

┌─arrayDistinct([1, 2, 2, 3, 1])─┐

│ [1,2,3] │

└────────────────────────────────┘

###### arrayEnumerateDense(arr)

Возвращает массив того же размера, что и исходный массив, с индексами исходного массива, указывающими, где каждый элемент впервые появляется в исходном массиве.

Пример:

**SELECT** arrayEnumerateDense([10, 20, 10, 30])

┌─arrayEnumerateDense([10, 20, 10, 30])─┐

│ [1,2,1,3] │

└───────────────────────────────────────┘

###### arrayIntersect(arr)

Принимает несколько массивов, возвращает массив с элементами, присутствующими во всех исходных массивах. Элементы на выходе следуют в порядке следования в первом массиве.

Пример:

**SELECT**

arrayIntersect([1, 2], [1, 3], [2, 3]) **AS** no\_intersect,

arrayIntersect([1, 2], [1, 3], [1, 4]) **AS** **intersect**

┌─no\_intersect─┬─intersect─┐

│ [] │ [1] │

└──────────────┴───────────┘

###### arrayReduce

Применяет агрегатную функцию к элементам массива и возвращает ее результат. Имя агрегирующей функции передается как строка в одинарных кавычках 'max', 'sum'. При использовании параметрических агрегатных функций, параметр указывается после имени функции в круглых скобках 'uniqUpTo(6)'.

Синтаксис:

arrayReduce(agg\_func, arr1, arr2, ..., arrN)

Параметры:

* agg\_func – имя агрегатной функции, которая должна быть константой [string](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).
* arr – любое количество столбцов типа [array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/) в качестве параметров агрегатной функции.

Возвращаемое значение:

Пример:

Запрос:

**SELECT** arrayReduce('max', [1, 2, 3])

Ответ:

┌─arrayReduce('max', [1, 2, 3])─┐

│ 3 │

└───────────────────────────────┘

Если агрегатная функция имеет несколько аргументов, то эту функцию можно применять к нескольким массивам одинакового размера.

Пример:

Запрос:

**SELECT** arrayReduce('maxIf', [3, 5], [1, 0])

Ответ:

┌─arrayReduce('maxIf', [3, 5], [1, 0])─┐

│ 3 │

└──────────────────────────────────────┘

Пример с параметрической агрегатной функцией:

Запрос:

**SELECT** arrayReduce('uniqUpTo(3)', [1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

Ответ:

┌─arrayReduce('uniqUpTo(3)', [1, 2, 3, 4, 5, 6, 7, 8, 9, 10])─┐

│ 4 │

└─────────────────────────────────────────────────────────────┘

###### arrayReduceInRanges

Применяет агрегатную функцию к элементам массива в заданных диапазонах и возвращает массив, содержащий результат, соответствующий каждому диапазону. Функция вернет тот же результат, что и несколько arrayReduce(agg\_func, arraySlice(arr1, index, length), ...).

Синтаксис:

arrayReduceInRanges(agg\_func, ranges, arr1, arr2, ..., arrN)

Параметры:

* agg\_func – имя агрегатной функции, которая должна быть [строковой](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/) константой.
* ranges – диапазоны для агрегирования, которые должны быть [массивом](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/) of [кортежей](https://clickhouse.tech/docs/ru/sql-reference/data-types/tuple/) который содержит индекс и длину каждого диапазона.
* arr – любое количество столбцов типа [Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/) в качестве параметров агрегатной функции.

Возвращаемое значение:

* Массив, содержащий результаты агрегатной функции для указанных диапазонов.

Тип: [Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/).

Пример:

Запрос:

**SELECT** arrayReduceInRanges(

'sum',

[(1, 5), (2, 3), (3, 4), (4, 4)],

[1000000, 200000, 30000, 4000, 500, 60, 7]

) **AS** res

Ответ:

┌─res─────────────────────────┐

│ [1234500,234000,34560,4567] │

└─────────────────────────────┘

###### arrayReverse(arr)

Возвращает массив того же размера, что и исходный массив, содержащий элементы в обратном порядке.

Пример:

**SELECT** arrayReverse([1, 2, 3])

┌─arrayReverse([1, 2, 3])─┐

│ [3,2,1] │

└─────────────────────────┘

###### reverse(arr)

Синоним для [«arrayReverse»](https://clickhouse.tech/docs/ru/sql-reference/functions/array-functions/#arrayreverse).

###### arrayFlatten

Преобразует массив массивов в плоский массив.

Функция:

* Оперирует с массивами любой вложенности.
* Не изменяет массив, если он уже плоский.

Результирующий массив содержит все элементы исходных массивов.

Синтаксис:

flatten(array\_of\_arrays)

Синоним: flatten.

Параметры:

* array\_of\_arrays – [массив](https://clickhouse.tech/docs/ru/sql-reference/functions/array-functions/) массивов. Например, [[1,2,3], [4,5]].

Примеры:

**SELECT** flatten([[[1]], [[2], [3]]])

┌─flatten(array(array([1]), array([2], [3])))─┐

│ [1,2,3] │

└─────────────────────────────────────────────┘

###### arrayCompact

Удаляет последовательно повторяющиеся элементы из массива. Порядок результирующих значений определяется порядком в исходном массиве.

Синтаксис:

arrayCompact(arr)

Параметры:

* arr – [массив](https://clickhouse.tech/docs/ru/sql-reference/functions/array-functions/) для обхода.

Возвращаемое значение:

Массив без последовательных дубликатов.

Тип: Array.

Пример:

Запрос:

**SELECT** arrayCompact([1, 1, nan, nan, 2, 3, 3, 3])

Ответ:

┌─arrayCompact([1, 1, nan, nan, 2, 3, 3, 3])─┐

│ [1,nan,nan,2,3] │

└────────────────────────────────────────────┘

###### arrayZip

Объединяет несколько массивов в один. Результирующий массив содержит соответственные элементы исходных массивов, сгруппированные в кортежи в указанном порядке аргументов.

Синтаксис:

arrayZip(arr1, arr2, ..., arrN)

Параметры:

* arrN – [массив](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/).

Функция принимает любое количество массивов, которые могут быть различных типов. Все массивы должны иметь одинаковую длину.

Возвращаемое значение:

* Массив с элементами исходных массивов, сгруппированными в [кортежи](https://clickhouse.tech/docs/ru/sql-reference/data-types/tuple/). Типы данных в кортежах соответствуют типам данных входных массивов и следуют в том же порядке, в котором переданы массивы.

Тип: [Массив](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/).

Пример:

Запрос:

**SELECT** arrayZip(['a', 'b', 'c'], [5, 2, 1])

Ответ:

┌─arrayZip(['a', 'b', 'c'], [5, 2, 1])─┐

│ [('a',5),('b',2),('c',1)] │

└──────────────────────────────────────┘

###### arrayMap(func, arr1, …)

Возвращает массив, полученный на основе результатов применения функции func к каждому элементу массива arr.

Примеры:

**SELECT** arrayMap(x -> (x + 2), [1, 2, 3]) **as** res;

┌─res─────┐

│ [3,4,5] │

└─────────┘

Следующий пример показывает, как создать кортежи из элементов разных массивов:

**SELECT** arrayMap((x, y) -> (x, y), [1, 2, 3], [4, 5, 6]) **AS** res

┌─res─────────────────┐

│ [(1,4),(2,5),(3,6)] │

└─────────────────────┘

Функция arrayMap является [функцией высшего порядка](https://clickhouse.tech/docs/ru/sql-reference/functions/#higher-order-functions) – в качестве первого аргумента ей нужно передать лямбда-функцию, и этот аргумент не может быть опущен.

###### arrayFilter(func, arr1, …)

Возвращает массив, содержащий только те элементы массива arr1, для которых функция func возвращает не 0.

Примеры:

**SELECT** arrayFilter(x -> x **LIKE** '%World%', ['Hello', 'abc World']) **AS** res

┌─res───────────┐

│ ['abc World'] │

└───────────────┘

**SELECT**

arrayFilter(

(i, x) -> x **LIKE** '%World%',

arrayEnumerate(arr),

['Hello', 'abc World'] **AS** arr)

**AS** res

┌─res─┐

│ [2] │

└─────┘

Функция arrayFilter является [функцией высшего порядка](https://clickhouse.tech/docs/ru/sql-reference/functions/#higher-order-functions) – в качестве первого аргумента ей нужно передать лямбда-функцию, и этот аргумент не может быть опущен.

###### arrayExists([func,] arr1, …)

Возвращает 1, если существует хотя бы один элемент массива arr, для которого функция func возвращает не 0. Иначе возвращает 0.

Функция arrayExists является [функцией высшего порядка](https://clickhouse.tech/docs/ru/sql-reference/functions/#higher-order-functions) – в качестве первого аргумента ей можно передать лямбда-функцию.

###### arrayAll([func,] arr1, …)

Возвращает 1, если для всех элементов массива arr, функция func возвращает не 0. Иначе возвращает 0.

Функция arrayAll является [функцией высшего порядка](https://clickhouse.tech/docs/ru/sql-reference/functions/#higher-order-functions) - в качестве первого аргумента ей можно передать лямбда-функцию.

###### arrayFirst(func, arr1, …)

Возвращает первый элемент массива arr1, для которого функция func возвращает не 0.

Функция arrayFirst является [функцией высшего порядка](https://clickhouse.tech/docs/ru/sql-reference/functions/#higher-order-functions) – в качестве первого аргумента ей нужно передать лямбда-функцию, и этот аргумент не может быть опущен.

###### arrayFirstIndex(func, arr1, …)

Возвращает индекс первого элемента массива arr1, для которого функция func возвращает не 0.

Функция arrayFirstIndex является [функцией высшего порядка](https://clickhouse.tech/docs/ru/sql-reference/functions/#higher-order-functions) – в качестве первого аргумента ей нужно передать лямбда-функцию, и этот аргумент не может быть опущен.

###### arrayMin

Возвращает значение минимального элемента в исходном массиве.

Если передана функция func, возвращается минимум из элементов массива, преобразованных этой функцией.

Функция arrayMin является [функцией высшего порядка](https://clickhouse.tech/docs/ru/sql-reference/functions/#higher-order-functions) – в качестве первого аргумента ей можно передать лямбда-функцию.

Синтаксис:

arrayMin([func,] arr)

Параметры:

* func – функция. [Expression](https://clickhouse.tech/docs/ru/sql-reference/data-types/special-data-types/expression/).
* arr – массив. [Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/).

Возвращаемое значение:

* Минимальное значение функции (или минимальный элемент массива).

Тип: если передана func, соответствует типу ее возвращаемого значения, иначе соответствует типу элементов массива.

Примеры:

Запрос:

**SELECT** arrayMin([1, 2, 4]) **AS** res;

Результат:

┌─res─┐

│ 1 │

└─────┘

Запрос:

**SELECT** arrayMin(x -> (-x), [1, 2, 4]) **AS** res;

Результат:

┌─res─┐

│ -4 │

└─────┘

###### arrayMax

Возвращает значение максимального элемента в исходном массиве.

Если передана функция func, возвращается максимум из элементов массива, преобразованных этой функцией.

Функция arrayMax является [функцией высшего порядка](https://clickhouse.tech/docs/ru/sql-reference/functions/#higher-order-functions) – в качестве первого аргумента ей можно передать лямбда-функцию.

Синтаксис:

arrayMax([func,] arr)

Параметры:

* func – функция. [Expression](https://clickhouse.tech/docs/ru/sql-reference/data-types/special-data-types/expression/).
* arr – массив. [Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/).

Возвращаемое значение

* Максимальное значение функции (или максимальный элемент массива).

Тип: если передана func, соответствует типу ее возвращаемого значения, иначе соответствует типу элементов массива.

Примеры:

Запрос:

**SELECT** arrayMax([1, 2, 4]) **AS** res;

Результат:

┌─res─┐

│ 4 │

└─────┘

Запрос:

**SELECT** arrayMax(x -> (-x), [1, 2, 4]) **AS** res;

Результат:

┌─res─┐

│ -1 │

└─────┘

###### arraySum

Возвращает сумму элементов в исходном массиве.

Если передана функция func, возвращается сумма элементов массива, преобразованных этой функцией.

Функция arraySum является [функцией высшего порядка](https://clickhouse.tech/docs/ru/sql-reference/functions/#higher-order-functions) – в качестве первого аргумента ей можно передать лямбда-функцию.

Синтаксис:

arraySum([func,] arr)

Параметры:

* func – функция. [Expression](https://clickhouse.tech/docs/ru/sql-reference/data-types/special-data-types/expression/).
* arr – массив. [Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/).

Возвращаемое значение:

* Сумма значений функции (или сумма элементов массива).

Тип: для Decimal чисел в исходном массиве (если функция func была передана, то для чисел, преобразованных ею) – [Decimal128](https://clickhouse.tech/docs/ru/sql-reference/data-types/decimal/), для чисел с плавающей точкой – [Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/), для беззнаковых целых чисел – [UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/), для целых чисел со знаком – [Int64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Примеры:

Запрос:

**SELECT** arraySum([2, 3]) **AS** res;

Результат:

┌─res─┐

│ 5 │

└─────┘

Запрос:

**SELECT** arraySum(x -> x\*x, [2, 3]) **AS** res;

Результат:

┌─res─┐

│ 13 │

└─────┘

###### arrayAvg

Возвращает среднее значение элементов в исходном массиве.

Если передана функция func, возвращается среднее значение элементов массива, преобразованных этой функцией.

Функция arrayAvg является [функцией высшего порядка](https://clickhouse.tech/docs/ru/sql-reference/functions/#higher-order-functions) – в качестве первого аргумента ей можно передать лямбда-функцию.

Синтаксис:

arrayAvg([func,] arr)

Параметры:

* func – функция. [Expression](https://clickhouse.tech/docs/ru/sql-reference/data-types/special-data-types/expression/).
* arr – массив. [Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/).

Возвращаемое значение:

* Среднее значение функции (или среднее значение элементов массива).

Тип: [Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/).

Примеры:

Запрос:

**SELECT** arrayAvg([1, 2, 4]) **AS** res;

Результат:

┌────────────────res─┐

│ 2.3333333333333335 │

└────────────────────┘

Запрос:

**SELECT** arrayAvg(x -> (x \* x), [2, 4]) **AS** res;

Результат:

┌─res─┐

│ 10 │

└─────┘

Синтаксис:

arraySum(arr)

Возвращаемое значение:

* Число.

Тип: [Int](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/) или [Float](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/).

Параметры:

* arr – [массив](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/).

Примеры:

Запрос:

**SELECT** arraySum([2,3]) **AS** res;

Результат:

┌─res─┐

│ 5 │

└─────┘

Запрос:

**SELECT** arraySum(x -> x\*x, [2, 3]) **AS** res;

Результат:

┌─res─┐

│ 13 │

└─────┘

###### arrayCumSum([func,] arr1, …)

Возвращает массив из частичных сумм элементов исходного массива (сумма с накоплением). Если указана функция func, то значения элементов массива преобразуются этой функцией перед суммированием.

Функция arrayCumSum является [функцией высшего порядка](https://clickhouse.tech/docs/ru/sql-reference/functions/#higher-order-functions) – в качестве первого аргумента ей можно передать лямбда-функцию.

Пример:

**SELECT** arrayCumSum([1, 1, 1, 1]) **AS** res

┌─res──────────┐

│ [1, 2, 3, 4] │

└──────────────┘

###### arrayAUC

Вычисляет площадь под кривой.

Синтаксис:

arrayAUC(arr\_scores, arr\_labels)

Параметры:

* arr\_scores – оценка, которую дает модель предсказания.
* arr\_labels – ярлыки выборок, обычно 1 для содержательных выборок и 0 для бессодержательных выборок.

Возвращаемое значение:

* Значение площади под кривой.

Тип данных: Float64.

Пример:

Запрос:

**select** arrayAUC([0.1, 0.4, 0.35, 0.8], [0, 0, 1, 1])

Ответ:

┌─arrayAUC([0.1, 0.4, 0.35, 0.8], [0, 0, 1, 1])─┐

│ 0.75 │

└────────────────────────────────────────---──┘

##### Функции сравнения

Функции сравнения возвращают всегда 0 или 1 (UInt8).

Сравнивать можно следующие типы:

* числа,
* строки и фиксированные строки,
* даты,
* даты-с-временем,
* внутри каждой группы, но не из разных групп.

Например, вы не можете сравнить дату со строкой. Надо использовать функцию преобразования строки в дату или наоборот.

Строки сравниваются побайтово. Более короткая строка меньше всех строк, начинающихся с неё и содержащих ещё хотя бы один символ.

Замечание. До версии 1.1.54134 сравнение знаковых и беззнаковых целых чисел производилось также, как в C++. То есть, вы могли получить неверный результат в таких случаях: SELECT 9223372036854775807 > -1. С версии 1.1.54134 поведение изменилось и стало математически корректным.

###### equals, оператор a = b и a == b

###### notEquals, оператор a != b и a <> b

###### less, оператор <

###### greater, оператор >

###### lessOrEquals, оператор <=

###### greaterOrEquals, оператор >=

##### Логические функции

Логические функции принимают любые числовые типы, а возвращают число типа UInt8, равное 0 или 1.

Ноль в качестве аргумента считается «ложью», а любое ненулевое значение - «истиной»:

* and, оператор AND,
* or, оператор OR,
* not, оператор NOT,
* xor.

##### Функции преобразования типов

###### Общие проблемы преобразования чисел

При преобразовании значения из одного типа в другой необходимо помнить, что в общем случае это небезопасная операция, которая может привести к потере данных. Потеря данных может произойти при попытке сконвертировать тип данных значения от большего к меньшему или при конвертировании между различными классами типов данных.

Поведение RT.WideStore при конвертировании похоже на [поведение C++ программ](https://en.cppreference.com/w/cpp/language/implicit_conversion).

###### toInt(8|16|32|64|128|256)

Преобразует входное значение к типу [Int](https://clickhouse.tech/docs/ru/sql-reference/functions/type-conversion-functions/). Семейство функций включает:

* toInt8(expr) – возвращает значение типа Int8.
* toInt16(expr) – возвращает значение типа Int16.
* toInt32(expr) – возвращает значение типа Int32.
* toInt64(expr) – возвращает значение типа Int64.
* toInt128(expr) – возвращает значение типа Int128.
* toInt256(expr) – возвращает значение типа Int256.

Параметры:

* expr – [выражение](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-expressions) возвращающее число или строку с десятичным представление числа. Бинарное, восьмеричное и шестнадцатеричное представление числа не поддержаны. Ведущие нули обрезаются.

Возвращаемое значение:

* Целое число типа Int8, Int16, Int32, Int64, Int128 или Int256.

Функции используют [округление к нулю](https://en.wikipedia.org/wiki/Rounding#Rounding_towards_zero), т.е. обрезают дробную часть числа.

Поведение функций для аргументов [NaN и Inf](https://clickhouse.tech/docs/ru/sql-reference/functions/type-conversion-functions/#data_type-float-nan-inf) не определено. При использовании функций помните о возможных проблемах при [преобразовании чисел](https://clickhouse.tech/docs/ru/sql-reference/functions/type-conversion-functions/#numeric-conversion-issues).

Пример:

Запрос:

**SELECT** toInt64(nan), toInt32(32), toInt16('16'), toInt8(8.8);

Результат:

┌─────────toInt64(nan)─┬─toInt32(32)─┬─toInt16('16')─┬─toInt8(8.8)─┐

│ -9223372036854775808 │ 32 │ 16 │ 8 │

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

###### toInt(8|16|32|64|128|256)OrZero

Принимает аргумент типа String и пытается его распарсить в Int(8|16|32|64|128|256). Если не удалось – возвращает 0.

Пример:

Запрос:

**SELECT** toInt64OrZero('123123'), toInt8OrZero('123qwe123');

Результат:

┌─toInt64OrZero('123123')─┬─toInt8OrZero('123qwe123')─┐

│ 123123 │ 0 │

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

###### toInt(8|16|32|64|128|256)OrNull

Принимает аргумент типа String и пытается его распарсить в Int(8|16|32|64|128|256). Если не удалось - возвращает NULL.

Пример:

Запрос:

**SELECT** toInt64OrNull('123123'), toInt8OrNull('123qwe123');

Результат:

┌─toInt64OrNull('123123')─┬─toInt8OrNull('123qwe123')─┐

│ 123123 │ ᴺᵁᴸᴸ │

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

###### toUInt(8|16|32|64|256)

Преобраует входное значение к типу [UInt](https://clickhouse.tech/docs/ru/sql-reference/functions/type-conversion-functions/). Семейство функций включает:

* toUInt8(expr) – возвращает значение типа UInt8.
* toUInt16(expr) – возвращает значение типа UInt16.
* toUInt32(expr) – возвращает значение типа UInt32.
* toUInt64(expr) – возвращает значение типа UInt64.
* toUInt256(expr) – возвращает значение типа UInt256.

Параметры:

* expr – [выражение](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-expressions) возвращающее число или строку с десятичным представление числа. Бинарное, восьмеричное и шестнадцатеричное представление числа не поддержаны. Ведущие нули обрезаются.

Возвращаемое значение:

* Целое число типа UInt8, UInt16, UInt32, UInt64 или UInt256.

Функции используют [округление к нулю](https://en.wikipedia.org/wiki/Rounding#Rounding_towards_zero), т.е. обрезают дробную часть числа.

Поведение функций для аргументов [NaN и Inf](https://clickhouse.tech/docs/ru/sql-reference/functions/type-conversion-functions/#data_type-float-nan-inf) не определено. Если передать строку, содержащую отрицательное число, например '-32', RT.WideStore генерирует исключение. При использовании функций помните о возможных проблемах при [преобразовании чисел](https://clickhouse.tech/docs/ru/sql-reference/functions/type-conversion-functions/#numeric-conversion-issues).

Пример:

Запрос:

**SELECT** toUInt64(nan), toUInt32(-32), toUInt16('16'), toUInt8(8.8);

Результат:

┌───────toUInt64(nan)─┬─toUInt32(-32)─┬─toUInt16('16')─┬─toUInt8(8.8)─┐

│ 9223372036854775808 │ 4294967264 │ 16 │ 8 │

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

###### toUInt(8|16|32|64|256)OrZero

###### toUInt(8|16|32|64|256)OrNull

###### toFloat(32|64)

###### toFloat(32|64)OrZero

###### toFloat(32|64)OrNull

###### toDate

Cиноним: DATE.

###### toDateOrZero

###### toDateOrNull

###### toDateTime

###### toDateTimeOrZero

###### toDateTimeOrNull

###### toDecimal(32|64|128|256)

Преобразует value к типу данных [Decimal](https://clickhouse.tech/docs/ru/sql-reference/functions/type-conversion-functions/) с точностью S. value может быть числом или строкой. Параметр S (scale) задаёт число десятичных знаков:

* toDecimal32(value, S).
* toDecimal64(value, S).
* toDecimal128(value, S).
* toDecimal256(value, S).

###### toDecimal(32|64|128|256)OrNull

Преобразует входную строку в значение с типом данных [Nullable (Decimal (P, S))](https://clickhouse.tech/docs/ru/sql-reference/functions/type-conversion-functions/). Семейство функций включает в себя:

* toDecimal32OrNull(expr, S) – возвращает значение типа Nullable(Decimal32(S)).
* toDecimal64OrNull(expr, S) – возвращает значение типа Nullable(Decimal64(S)).
* toDecimal128OrNull(expr, S) – возвращает значение типа Nullable(Decimal128(S)).
* toDecimal256OrNull(expr, S) – возвращает значение типа Nullable(Decimal256(S)).

Эти функции следует использовать вместо функций toDecimal\*(), если при ошибке обработки входного значения вы хотите получать NULL вместо исключения.

Параметры:

* expr – [выражение](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-expressions), возвращающее значение типа [String](https://clickhouse.tech/docs/ru/sql-reference/functions/type-conversion-functions/). RT.WideStore ожидает текстовое представление десятичного числа. Например, '1.111'.
* S – количество десятичных знаков в результирующем значении.

Возвращаемое значение:

* Значение типа Nullable(Decimal(P,S)).

Значение содержит:

* Число с S десятичными знаками, если RT.WideStore распознал число во входной строке.
* NULL, если RT.WideStore не смог распознать число во входной строке или входное число содержит больше чем S десятичных знаков.

Примеры:

Запрос:

**SELECT** toDecimal32OrNull(toString(-1.111), 5) **AS** val, toTypeName(val);

Результат:

┌──────val─┬─toTypeName(toDecimal32OrNull(toString(-1.111), 5))─┐

│ -1.11100 │ Nullable(Decimal(9, 5)) │

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

Запрос:

**SELECT** toDecimal32OrNull(toString(-1.111), 2) **AS** val, toTypeName(val);

Результат:

┌──val─┬─toTypeName(toDecimal32OrNull(toString(-1.111), 2))─┐

│ ᴺᵁᴸᴸ │ Nullable(Decimal(9, 2)) │

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

###### toDecimal(32|64|128|256)OrZero

Преобразует тип входного значения в [Decimal (P, S)](https://clickhouse.tech/docs/ru/sql-reference/functions/type-conversion-functions/). Семейство функций включает в себя:

* toDecimal32OrZero( expr, S) – возвращает значение типа Decimal32(S).
* toDecimal64OrZero( expr, S) – возвращает значение типа Decimal64(S).
* toDecimal128OrZero( expr, S) – возвращает значение типа Decimal128(S).
* toDecimal256OrZero( expr, S) – возвращает значение типа Decimal256(S).

Эти функции следует использовать вместо функций toDecimal\*(), если при ошибке обработки входного значения вы хотите получать 0 вместо исключения.

Параметры

* expr – [выражение](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-expressions), возвращающее значение типа [String](https://clickhouse.tech/docs/ru/sql-reference/functions/type-conversion-functions/). RT.WideStore ожидает текстовое представление десятичного числа. Например, '1.111'.
* S – количество десятичных знаков в результирующем значении.

Возвращаемое значение

Значение типа Nullable(Decimal(P,S)). P равно числовой части имени функции. Например, для функции toDecimal32OrZero, P = 32. Значение содержит:

* Число с S десятичными знаками, если RT.WideStore распознал число во входной строке.
* 0 c S десятичными знаками, если RT.WideStore не смог распознать число во входной строке или входное число содержит больше чем S десятичных знаков.

Пример:

Запрос:

**SELECT** toDecimal32OrZero(toString(-1.111), 5) **AS** val, toTypeName(val);

Результат:

┌──────val─┬─toTypeName(toDecimal32OrZero(toString(-1.111), 5))─┐

│ -1.11100 │ Decimal(9, 5) │

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

Запрос:

**SELECT** toDecimal32OrZero(toString(-1.111), 2) **AS** val, toTypeName(val);

Результат:

┌──val─┬─toTypeName(toDecimal32OrZero(toString(-1.111), 2))─┐

│ 0.00 │ Decimal(9, 2) │

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

###### toString

Функции преобразования между числами, строками (но не фиксированными строками), датами и датами-с-временем.

Все эти функции принимают один аргумент.

При преобразовании в строку или из строки, производится форматирование или парсинг значения по тем же правилам, что и для формата TabSeparated (и почти всех остальных текстовых форматов). Если распарсить строку не удаётся - кидается исключение и выполнение запроса прерывается.

При преобразовании даты в число или наоборот, дате соответствует число дней от начала unix эпохи.

При преобразовании даты-с-временем в число или наоборот, дате-с-временем соответствует число секунд от начала unix эпохи.

Форматы даты и даты-с-временем для функций toDate/toDateTime определены следующим образом:

YYYY-MM-DD

YYYY-MM-DD hh:mm:ss

В качестве исключения, если делается преобразование из числа типа UInt32, Int32, UInt64, Int64 в Date, и если число больше или равно 65536, то число рассматривается как unix timestamp (а не как число дней) и округляется до даты. Это позволяет поддержать распространённый случай, когда пишут toDate(unix\_timestamp), что иначе было бы ошибкой и требовало бы написания более громоздкого toDate(toDateTime(unix\_timestamp))

Преобразование между датой и датой-с-временем производится естественным образом: добавлением нулевого времени или отбрасыванием времени.

Преобразование между числовыми типами производится по тем же правилам, что и присваивание между разными числовыми типами в C++.

Дополнительно, функция toString от аргумента типа DateTime может принимать второй аргумент String - имя тайм-зоны. Пример: Asia/Yekaterinburg в этом случае, форматирование времени производится согласно указанной тайм-зоне.

Пример:

Запрос:

**SELECT**

now() **AS** now\_local,

toString(now(), 'Asia/Yekaterinburg') **AS** now\_yekat;

Результат:

┌───────────now\_local─┬─now\_yekat───────────┐

│ 2016-06-15 00:11:21 │ 2016-06-15 02:11:21 │

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

Также смотрите функцию toUnixTimestamp.

###### toFixedString(s, N)

Преобразует аргумент типа String в тип FixedString(N) (строку фиксированной длины N). N должно быть константой.

Если строка имеет меньше байт, чем N, то она дополняется нулевыми байтами справа. Если строка имеет больше байт, чем N - кидается исключение.

###### toStringCutToZero(s)

Принимает аргумент типа String или FixedString. Возвращает String, вырезая содержимое строки до первого найденного нулевого байта.

Примеры:

Запрос:

**SELECT** toFixedString('foo', 8) **AS** s, toStringCutToZero(s) **AS** s\_cut;

Результат:

┌─s─────────────┬─s\_cut─┐

│ foo\0\0\0\0\0 │ foo │

└───────────────┴───────┘

Запрос:

**SELECT** toFixedString('foo\0bar', 8) **AS** s, toStringCutToZero(s) **AS** s\_cut;

Результат:

┌─s──────────┬─s\_cut─┐

│ foo\0bar\0 │ foo │

└────────────┴───────┘

###### reinterpretAsUInt(8|16|32|64)

###### reinterpretAsInt(8|16|32|64)

###### reinterpretAsFloat(32|64)

###### reinterpretAsDate

###### reinterpretAsDateTime

Функции принимают строку и интерпретируют байты, расположенные в начале строки, как число в host order (little endian). Если строка имеет недостаточную длину, то функции работают так, как будто строка дополнена необходимым количеством нулевых байт. Если строка длиннее, чем нужно, то лишние байты игнорируются. Дата интерпретируется, как число дней с начала unix-эпохи, а дата-с-временем – как число секунд с начала unix-эпохи.

###### reinterpretAsString

Функция принимает число или дату, или дату-с-временем и возвращает строку, содержащую байты, представляющие соответствующее значение в host order (little endian). При этом, отбрасываются нулевые байты с конца. Например, значение 255 типа UInt32 будет строкой длины 1 байт.

###### reinterpretAsUUID

Функция принимает шестнадцатибайтную строку и интерпретирует ее байты в network order (big-endian). Если строка имеет недостаточную длину, то функция работает так, как будто строка дополнена необходимым количетсвом нулевых байт с конца. Если строка длиннее, чем шестнадцать байт, то игнорируются лишние байты с конца.

Синтаксис:

reinterpretAsUUID(fixed\_string)

Параметры:

* fixed\_string – cтрока с big-endian порядком байтов. [FixedString](https://clickhouse.tech/docs/ru/sql-reference/data-types/fixedstring/#fixedstring).

Возвращаемое значение:

* Значение типа [UUID](https://clickhouse.tech/docs/ru/sql-reference/data-types/uuid/#uuid-data-type).

Примеры:

Интерпретация строки как UUID.

Запрос:

**SELECT** reinterpretAsUUID(reverse(unhex('000102030405060708090a0b0c0d0e0f')));

Результат:

┌─reinterpretAsUUID(reverse(unhex('000102030405060708090a0b0c0d0e0f')))─┐

│ 08090a0b-0c0d-0e0f-0001-020304050607 │

└───────────────────────────────────────────────────────────────────────┘

Переход в UUID и обратно.

Запрос:

**WITH**

generateUUIDv4() **AS** uuid,

**identity**(**lower**(hex(reverse(reinterpretAsString(uuid))))) **AS** str,

reinterpretAsUUID(reverse(unhex(str))) **AS** uuid2

**SELECT** uuid = uuid2;

Результат:

┌─equals(uuid, uuid2)─┐

│ 1 │

└─────────────────────┘

###### CAST(x, T)

Преобразует входное значение x в указанный тип данных T.

Поддерживается также синтаксис CAST(x AS t).

Обратите внимание, что если значение x не может быть преобразовано к типу T, возникает переполнение. Например, CAST(-1, 'UInt8') возвращает 255.

Пример:

Запрос:

**SELECT**

'2016-06-15 23:00:00' **AS** **timestamp**,

**CAST**(**timestamp** **AS** DateTime) **AS** datetime,

**CAST**(**timestamp** **AS** Date) **AS** date,

**CAST**(**timestamp**, 'String') **AS** string,

**CAST**(**timestamp**, 'FixedString(22)') **AS** fixed\_string;

Результат:

┌─timestamp───────────┬────────────datetime─┬───────date─┬─string──────────────┬─fixed\_string──────────────┐

│ 2016-06-15 23:00:00 │ 2016-06-15 23:00:00 │ 2016-06-15 │ 2016-06-15 23:00:00 │ 2016-06-15 23:00:00\0\0\0 │

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

Преобразование в FixedString(N) работает только для аргументов типа String или FixedString(N).

Поддержано преобразование к типу [Nullable](https://clickhouse.tech/docs/ru/sql-reference/functions/type-conversion-functions/) и обратно.

Примеры:

Запрос:

**SELECT** toTypeName(x) **FROM** t\_null;

Результат:

┌─toTypeName(x)─┐

│ Int8 │

│ Int8 │

└───────────────┘

Запрос:

**SELECT** toTypeName(**CAST**(x, 'Nullable(UInt16)')) **FROM** t\_null;

Результат:

┌─toTypeName(CAST(x, 'Nullable(UInt16)'))─┐

│ Nullable(UInt16) │

│ Nullable(UInt16) │

└─────────────────────────────────────────┘

См. также:

* Настройка [cast\_keep\_nullable](https://clickhouse.tech/docs/ru/operations/settings/settings/#cast_keep_nullable).

###### accurateCast(x, T)

Преобразует входное значение x в указанный тип данных T.

В отличие от функции [cast(x, T)](https://clickhouse.tech/docs/ru/sql-reference/functions/type-conversion-functions/#type_conversion_function-cast), accurateCast не допускает переполнения при преобразовании числовых типов. Например, accurateCast(-1, 'UInt8') вызовет исключение.

Примеры:

Запрос:

**SELECT** **cast**(-1, 'UInt8') **as** uint8;

Результат:

┌─uint8─┐

│ 255 │

└─────

Запрос:

```sql

SELECT accurateCast(-1, 'UInt8') as uint8;

Результат:

Code: 70. DB::Exception: Received from localhost:9000. DB::Exception: Value in column Int8 cannot be safely converted into type UInt8: While processing accurateCast(-1, 'UInt8') AS uint8.

###### accurateCastOrNull(x, T)

Преобразует входное значение x в указанный тип данных T.

Всегда возвращает тип [Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/). Если исходное значение не может быть преобразовано к целевому типу, возвращает [NULL](https://clickhouse.tech/docs/ru/sql-reference/syntax/#null-literal).

Синтаксис:

accurateCastOrNull(x, T)

Параметры:

* x – входное значение.
* T – имя возвращаемого типа данных.

Возвращаемое значение:

* Значение, преобразованное в указанный тип T.

Примеры:

Запрос:

**SELECT** toTypeName(accurateCastOrNull(5, 'UInt8'));

Результат:

┌─toTypeName(accurateCastOrNull(5, 'UInt8'))─┐

│ Nullable(UInt8) │

└────────────────────────────────────────────┘

Запрос:

**SELECT**

accurateCastOrNull(-1, 'UInt8') **as** uint8,

accurateCastOrNull(128, 'Int8') **as** int8,

accurateCastOrNull('Test', 'FixedString(2)') **as** fixed\_string;

Результат:

┌─uint8─┬─int8─┬─fixed\_string─┐

│ ᴺᵁᴸᴸ │ ᴺᵁᴸᴸ │ ᴺᵁᴸᴸ │

└───────┴──────┴──────────────┘

###### toInterval(Year|Quarter|Month|Week|Day|Hour|Minute|Second)

Приводит аргумент из числового типа данных к типу данных [IntervalType](https://clickhouse.tech/docs/ru/sql-reference/data-types/special-data-types/interval/).

Синтаксис:

toIntervalSecond(number)

toIntervalMinute(number)

toIntervalHour(number)

toIntervalDay(number)

toIntervalWeek(number)

toIntervalMonth(number)

toIntervalQuarter(number)

toIntervalYear(number)

Параметры:

* number – длительность интервала. Положительное целое число.

Возвращаемые значения:

* Значение с типом данных Interval.

Пример:

Запрос:

**WITH**

toDate('2019-01-01') **AS** date,

INTERVAL 1 WEEK **AS** interval\_week,

toIntervalWeek(1) **AS** interval\_to\_week

**SELECT**

date + interval\_week,

date + interval\_to\_week;

Результат:

┌─plus(date, interval\_week)─┬─plus(date, interval\_to\_week)─┐

│ 2019-01-08 │ 2019-01-08 │

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

###### parseDateTimeBestEffort

Преобразует дату и время в [строковом](https://clickhouse.tech/docs/ru/sql-reference/functions/type-conversion-functions/) представлении к типу данных [DateTime](https://clickhouse.tech/docs/ru/sql-reference/functions/type-conversion-functions/#data_type-datetime).

Функция распознаёт форматы [ISO 8601](https://en.wikipedia.org/wiki/ISO_8601), [RFC 1123 - 5.2.14 RFC-822 Date and Time Specification](https://tools.ietf.org/html/rfc1123#page-55), формат даты времени RT.WideStore’s а также некоторые другие форматы.

Синтаксис:

parseDateTimeBestEffort(time\_string[, time\_zone])

Параметры:

* time\_string – строка, содержащая дату и время для преобразования. [String](https://clickhouse.tech/docs/ru/sql-reference/functions/type-conversion-functions/).
* time\_zone – часовой пояс. Функция анализирует time\_string в соответствии с заданным часовым поясом. [String](https://clickhouse.tech/docs/ru/sql-reference/functions/type-conversion-functions/).

Поддерживаемые нестандартные форматы:

* [Unix timestamp](https://ru.wikipedia.org/wiki/Unix-%D0%B2%D1%80%D0%B5%D0%BC%D1%8F) в строковом представлении. 9 или 10 символов.
* Строка с датой и временем: YYYYMMDDhhmmss, DD/MM/YYYY hh:mm:ss, DD-MM-YY hh:mm, YYYY-MM-DD hh:mm:ss, etc.
* Строка с датой, но без времени: YYYY, YYYYMM, YYYY\*MM, DD/MM/YYYY, DD-MM-YY и т.д.
* Строка с временем, и с днём: DD, DD hh, DD hh:mm. В этом случае YYYY-MM принимается равным 2000-01.
* Строка, содержащая дату и время вместе с информацией о часовом поясе: YYYY-MM-DD hh:mm:ss ±h:mm, и т.д. Например, 2020-12-12 17:36:00 -5:00.

Для всех форматов с разделителями функция распознаёт названия месяцев, выраженных в виде полного англоязычного имени месяца или в виде первых трёх символов имени месяца. Примеры: 24/DEC/18, 24-Dec-18, 01-September-2018.

Возвращаемое значение:

* time\_string преобразованная к типу данных DateTime.

Примеры:

Запрос:

**SELECT** parseDateTimeBestEffort('12/12/2020 12:12:57')

**AS** parseDateTimeBestEffort;

Результат:

┌─parseDateTimeBestEffort─┐

│ 2020-12-12 12:12:57 │

└─────────────────────────┘

Запрос:

**SELECT** parseDateTimeBestEffort('Sat, 18 Aug 2018 07:22:16 GMT', 'Europe/Moscow')

**AS** parseDateTimeBestEffort;

Результат:

┌─parseDateTimeBestEffort─┐

│ 2018-08-18 10:22:16 │

└─────────────────────────┘

Запрос:

**SELECT** parseDateTimeBestEffort('1284101485')

**AS** parseDateTimeBestEffort;

Результат:

┌─parseDateTimeBestEffort─┐

│ 2015-07-07 12:04:41 │

└─────────────────────────┘

Запрос:

**SELECT** parseDateTimeBestEffort('2018-12-12 10:12:12')

**AS** parseDateTimeBestEffort;

Результат:

┌─parseDateTimeBestEffort─┐

│ 2018-12-12 10:12:12 │

└─────────────────────────┘

Запрос:

**SELECT** parseDateTimeBestEffort('10 20:19');

Результат:

┌─parseDateTimeBestEffort('10 20:19')─┐

│ 2000-01-10 20:19:00 │

└─────────────────────────────────────┘

См. также:

* [Информация о формате ISO 8601 от @xkcd](https://xkcd.com/1179/).
* [RFC 1123](https://tools.ietf.org/html/rfc1123).
* [toDate](https://clickhouse.tech/docs/ru/sql-reference/functions/type-conversion-functions/#todate).
* [toDateTime](https://clickhouse.tech/docs/ru/sql-reference/functions/type-conversion-functions/#todatetime).

###### parseDateTimeBestEffortUS

Эта функция похожа на [‘parseDateTimeBestEffort’](https://clickhouse.tech/docs/ru/sql-reference/functions/type-conversion-functions/#parsedatetimebesteffort), но разница состоит в том, что в она предполагает американский формат даты (MM/DD/YYYY etc.) в случае неоднозначности.

Синтаксис:

parseDateTimeBestEffortUS(time\_string [, time\_zone])

Параметры:

* time\_string – строка, содержащая дату и время для преобразования. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).
* time\_zone – часовой пояс. Функция анализирует time\_string в соответствии с часовым поясом. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).

Поддерживаемые нестандартные форматы:

* Строка, содержащая 9-10 цифр [unix timestamp](https://en.wikipedia.org/wiki/Unix_time).
* Строка, содержащая дату и время: YYYYMMDDhhmmss, MM/DD/YYYY hh:mm:ss, MM-DD-YY hh:mm, YYYY-MM-DD hh:mm:ss, etc.
* Строка с датой, но без времени: YYYY, YYYYMM, YYYY\*MM, MM/DD/YYYY, MM-DD-YY etc.
* Строка, содержащая день и время: DD, DD hh, DD hh:mm. В этом случае YYYY-MM заменяется на 2000-01.
* Строка, содержащая дату и время, а также информацию о часовом поясе: YYYY-MM-DD hh:mm:ss ±h:mm и т.д. Например, 2020-12-12 17:36:00 -5:00.

Возвращаемое значение:

* time\_string преобразован в тип данных DateTime.

Примеры:

Запрос:

**SELECT** parseDateTimeBestEffortUS('09/12/2020 12:12:57')

**AS** parseDateTimeBestEffortUS;

Результат:

┌─parseDateTimeBestEffortUS─┐

│ 2020-09-12 12:12:57 │

└─────────────────────────––┘

Запрос:

**SELECT** parseDateTimeBestEffortUS('09-12-2020 12:12:57')

**AS** parseDateTimeBestEffortUS;

Результат:

┌─parseDateTimeBestEffortUS─┐

│ 2020-09-12 12:12:57 │

└─────────────────────────––┘

Запрос:

**SELECT** parseDateTimeBestEffortUS('09.12.2020 12:12:57')

**AS** parseDateTimeBestEffortUS;

Результат:

┌─parseDateTimeBestEffortUS─┐

│ 2020-09-12 12:12:57 │

└─────────────────────────––┘

###### parseDateTimeBestEffortUSOrNull

Работает аналогично функции [parseDateTimeBestEffortUS](https://clickhouse.tech/docs/ru/sql-reference/functions/type-conversion-functions/#parsedatetimebesteffortUS), но в отличие от нее возвращает NULL, если входная строка не может быть преобразована в тип данных [DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/).

Синтаксис:

parseDateTimeBestEffortUSOrNull(time\_string[, time\_zone])

Параметры:

* time\_string – строка, содержащая дату или дату со временем для преобразования. Дата должна быть в американском формате (MM/DD/YYYY и т.д.). [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).
* time\_zone – [часовой пояс](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/#server_configuration_parameters-timezone). Функция анализирует time\_string в соответствии с заданным часовым поясом. Опциональный параметр. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).

Поддерживаемые нестандартные форматы:

* Строка в формате [unix timestamp](https://en.wikipedia.org/wiki/Unix_time), содержащая 9-10 цифр.
* Строка, содержащая дату и время: YYYYMMDDhhmmss, MM/DD/YYYY hh:mm:ss, MM-DD-YY hh:mm, YYYY-MM-DD hh:mm:ss и т.д.
* Строка, содержащая дату без времени: YYYY, YYYYMM, YYYY\*MM, MM/DD/YYYY, MM-DD-YY и т.д.
* Строка, содержащая день и время: DD, DD hh, DD hh:mm. В этом случае YYYY-MM заменяется на 2000-01.
* Строка, содержащая дату и время, а также информацию о часовом поясе: YYYY-MM-DD hh:mm:ss ±h:mm и т.д. Например, 2020-12-12 17:36:00 -5:00.

Возвращаемые значения:

* time\_string, преобразованная в тип данных DateTime.
* NULL, если входная строка не может быть преобразована в тип данных DateTime.

Примеры:

Запрос:

**SELECT** parseDateTimeBestEffortUSOrNull('02/10/2021 21:12:57') **AS** parseDateTimeBestEffortUSOrNull;

Результат:

┌─parseDateTimeBestEffortUSOrNull─┐

│ 2021-02-10 21:12:57 │

└─────────────────────────────────┘

Запрос:

**SELECT** parseDateTimeBestEffortUSOrNull('02-10-2021 21:12:57 GMT', 'Europe/Moscow') **AS** parseDateTimeBestEffortUSOrNull;

Результат:

┌─parseDateTimeBestEffortUSOrNull─┐

│ 2021-02-11 00:12:57 │

└─────────────────────────────────┘

Запрос:

**SELECT** parseDateTimeBestEffortUSOrNull('02.10.2021') **AS** parseDateTimeBestEffortUSOrNull;

Результат:

┌─parseDateTimeBestEffortUSOrNull─┐

│ 2021-02-10 00:00:00 │

└─────────────────────────────────┘

Запрос:

**SELECT** parseDateTimeBestEffortUSOrNull('10.2021') **AS** parseDateTimeBestEffortUSOrNull;

Результат:

┌─parseDateTimeBestEffortUSOrNull─┐

│ ᴺᵁᴸᴸ │

└─────────────────────────────────┘

###### parseDateTimeBestEffortUSOrZero

Работает аналогично функции [parseDateTimeBestEffortUS](https://clickhouse.tech/docs/ru/sql-reference/functions/type-conversion-functions/#parsedatetimebesteffortUS), но в отличие от нее возвращает нулевую дату (1970-01-01) или нулевую дату со временем (1970-01-01 00:00:00), если входная строка не может быть преобразована в тип данных [DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/).

Синтаксис:

parseDateTimeBestEffortUSOrZero(time\_string[, time\_zone])

Параметры:

* time\_string – строка, содержащая дату или дату со временем для преобразования. Дата должна быть в американском формате (MM/DD/YYYY и т.д.). [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).
* time\_zone – [часовой пояс](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/#server_configuration_parameters-timezone). Функция анализирует time\_string в соответствии с заданным часовым поясом. Опциональный параметр. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).

Поддерживаемые нестандартные форматы:

* Строка в формате [unix timestamp](https://en.wikipedia.org/wiki/Unix_time), содержащая 9-10 цифр.
* Строка, содержащая дату и время: YYYYMMDDhhmmss, MM/DD/YYYY hh:mm:ss, MM-DD-YY hh:mm, YYYY-MM-DD hh:mm:ss и т.д.
* Строка, содержащая дату без времени: YYYY, YYYYMM, YYYY\*MM, MM/DD/YYYY, MM-DD-YY и т.д.
* Строка, содержащая день и время: DD, DD hh, DD hh:mm. В этом случае YYYY-MM заменяется на 2000-01.
* Строка, содержащая дату и время, а также информацию о часовом поясе: YYYY-MM-DD hh:mm:ss ±h:mm и т.д. Например, 2020-12-12 17:36:00 -5:00.

Возвращаемые значения:

* time\_string, преобразованная в тип данных DateTime.
* Нулевая дата или нулевая дата со временем, если входная строка не может быть преобразована в тип данных DateTime.

Примеры:

Запрос:

**SELECT** parseDateTimeBestEffortUSOrZero('02/10/2021 21:12:57') **AS** parseDateTimeBestEffortUSOrZero;

Результат:

┌─parseDateTimeBestEffortUSOrZero─┐

│ 2021-02-10 21:12:57 │

└─────────────────────────────────┘

Запрос:

**SELECT** parseDateTimeBestEffortUSOrZero('02-10-2021 21:12:57 GMT', 'Europe/Moscow') **AS** parseDateTimeBestEffortUSOrZero;

Результат:

┌─parseDateTimeBestEffortUSOrZero─┐

│ 2021-02-11 00:12:57 │

└─────────────────────────────────┘

Запрос:

**SELECT** parseDateTimeBestEffortUSOrZero('02.10.2021') **AS** parseDateTimeBestEffortUSOrZero;

Результат:

┌─parseDateTimeBestEffortUSOrZero─┐

│ 2021-02-10 00:00:00 │

└─────────────────────────────────┘

Запрос:

**SELECT** parseDateTimeBestEffortUSOrZero('02.2021') **AS** parseDateTimeBestEffortUSOrZero;

Результат:

┌─parseDateTimeBestEffortUSOrZero─┐

│ 1970-01-01 00:00:00 │

└─────────────────────────────────┘

###### toUnixTimestamp64Milli

###### toUnixTimestamp64Micro

###### toUnixTimestamp64Nano

Преобразует значение DateTime64 в значение Int64 с фиксированной точностью менее одной секунды.

Входное значение округляется соответствующим образом вверх или вниз в зависимости от его точности. Обратите внимание, что возвращаемое значение – это временная метка в UTC, а не в часовом поясе DateTime64.

Синтаксис:

toUnixTimestamp64Milli(value)

Параметры:

* value – значение DateTime64 с любой точностью.

Возвращаемое значение:

* Значение value, преобразованное в тип данных Int64.

Примеры:

Запрос:

**WITH** toDateTime64('2019-09-16 19:20:12.345678910', 6) **AS** dt64

**SELECT** toUnixTimestamp64Milli(dt64);

Результат:

┌─toUnixTimestamp64Milli(dt64)─┐

│ 1568650812345 │

└──────────────────────────────┘

Запрос:

**WITH** toDateTime64('2019-09-16 19:20:12.345678910', 6) **AS** dt64

**SELECT** toUnixTimestamp64Nano(dt64);

Результат:

┌─toUnixTimestamp64Nano(dt64)─┐

│ 1568650812345678000 │

└─────────────────────────────┘

###### fromUnixTimestamp64Milli

###### fromUnixTimestamp64Micro

###### fromUnixTimestamp64Nano

Преобразует значение Int64 в значение DateTime64 с фиксированной точностью менее одной секунды и дополнительным часовым поясом. Входное значение округляется соответствующим образом вверх или вниз в зависимости от его точности. Обратите внимание, что входное значение обрабатывается как метка времени UTC, а не метка времени в заданном (или неявном) часовом поясе.

Синтаксис:

fromUnixTimestamp64Milli(value [, ti])

Параметры:

* value – значение типы Int64 с любой точностью.
* timezone – (не обязательный параметр) часовой пояс в формате String для возвращаемого результата.

Возвращаемое значение:

* Значение value, преобразованное в тип данных DateTime64.

Пример:

Запрос:

**WITH** **CAST**(1234567891011, 'Int64') **AS** i64

**SELECT** fromUnixTimestamp64Milli(i64, 'UTC');

Результат:

┌─fromUnixTimestamp64Milli(i64, 'UTC')─┐

│ 2009-02-13 23:31:31.011 │

└──────────────────────────────────────┘

###### toLowCardinality

Преобразует входные данные в версию [LowCardianlity](https://clickhouse.tech/docs/ru/sql-reference/data-types/lowcardinality/) того же типа данных.

Чтобы преобразовать данные из типа LowCardinality, используйте функцию [CAST](https://clickhouse.tech/docs/ru/sql-reference/functions/type-conversion-functions/#type_conversion_function-cast). Например, CAST(x as String).

Синтаксис:

toLowCardinality(expr)

Параметры:

* expr – [выражение](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-expressions), которое в результате преобразуется в один из [поддерживаемых типов данных](https://clickhouse.tech/docs/ru/sql-reference/data-types/#data_types).

Возвращаемое значение:

* Результат преобразования expr.

Тип: LowCardinality(expr\_result\_type).

Пример:

Запрос:

**SELECT** toLowCardinality('1');

Результат:

┌─toLowCardinality('1')─┐

│ 1 │

└───────────────────────┘

###### formatRow

Преобразует произвольные выражения в строку заданного формата.

Синтаксис:

formatRow(format, x, y, ...)

Параметры:

* format – текстовый формат. Например, [CSV](https://clickhouse.tech/docs/ru/interfaces/formats/#csv), [TSV](https://clickhouse.tech/docs/ru/interfaces/formats/#tabseparated).
* x,y, ... – выражения.

Возвращаемое значение:

* Отформатированная строка (в текстовых форматах обычно с завершающим переводом строки).

Пример:

Запрос:

**SELECT** formatRow('CSV', number, 'good')

**FROM** numbers(3);

Результат:

┌─formatRow('CSV', number, 'good')─┐

│ 0,"good"

│

│ 1,"good"

│

│ 2,"good"

│

└──────────────────────────────────┘

###### formatRowNoNewline

Преобразует произвольные выражения в строку заданного формата. При этом удаляет лишние переводы строк \n, если они появились.

Синтаксис:

formatRowNoNewline(format, x, y, ...)

Параметры:

* format – текстовый формат. Например, [CSV](https://clickhouse.tech/docs/ru/interfaces/formats/#csv), [TSV](https://clickhouse.tech/docs/ru/interfaces/formats/#tabseparated).
* x,y, ... – выражения.
* Возвращаемое значение:
* Отформатированная строка (в текстовых форматах без завершающего перевода строки).

Пример:

Запрос:

**SELECT** formatRowNoNewline('CSV', number, 'good')

**FROM** numbers(3);

Результат:

┌─formatRowNoNewline('CSV', number, 'good')─┐

│ 0,"good" │

│ 1,"good" │

│ 2,"good" │

└───────────────────────────────────────────┘

##### Функции для работы с датами и временем

Поддержка часовых поясов.

Все функции по работе с датой и временем, для которых это имеет смысл, могут принимать второй, необязательный аргумент – имя часового пояса. Пример: Asia/Yekaterinburg. В этом случае, они используют не локальный часовой пояс (по умолчанию), а указанный:

**SELECT**

toDateTime('2016-06-15 23:00:00') **AS** time,

toDate(time) **AS** date\_local,

toDate(time, 'Asia/Yekaterinburg') **AS** date\_yekat,

toString(time, 'US/Samoa') **AS** time\_samoa

┌────────────────time─┬─date\_local─┬─date\_yekat─┬─time\_samoa──────────┐

│ 2016-06-15 23:00:00 │ 2016-06-15 │ 2016-06-16 │ 2016-06-15 09:00:00 │

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

###### toTimeZone

Переводит дату или дату-с-временем в указанный часовой пояс. Часовой пояс (таймзона) это атрибут типов Date/DateTime, внутреннее значение (количество секунд) поля таблицы или колонки результата не изменяется, изменяется тип поля и автоматически его текстовое отображение:

**SELECT**

toDateTime('2019-01-01 00:00:00', 'UTC') **AS** time\_utc,

toTypeName(time\_utc) **AS** type\_utc,

toInt32(time\_utc) **AS** int32utc,

toTimeZone(time\_utc, 'Asia/Yekaterinburg') **AS** time\_yekat,

toTypeName(time\_yekat) **AS** type\_yekat,

toInt32(time\_yekat) **AS** int32yekat,

toTimeZone(time\_utc, 'US/Samoa') **AS** time\_samoa,

toTypeName(time\_samoa) **AS** type\_samoa,

toInt32(time\_samoa) **AS** int32samoa

FORMAT Vertical;

Row 1:

──────

time\_utc: 2019-01-01 00:00:00

type\_utc: DateTime('UTC')

int32utc: 1546300800

time\_yekat: 2019-01-01 05:00:00

type\_yekat: DateTime('Asia/Yekaterinburg')

int32yekat: 1546300800

time\_samoa: 2018-12-31 13:00:00

type\_samoa: DateTime('US/Samoa')

int32samoa: 1546300800

toTimeZone(time\_utc, 'Asia/Yekaterinburg') изменяет тип DateTime('UTC') в DateTime('Asia/Yekaterinburg'). Значение (unix-время) 1546300800 остается неизменным, но текстовое отображение (результат функции toString()) меняется time\_utc: 2019-01-01 00:00:00 в time\_yekat: 2019-01-01 05:00:00.

###### toYear

Переводит дату или дату-с-временем в число типа UInt16, содержащее номер года (AD).

Синоним: YEAR.

###### toQuarter

Переводит дату или дату-с-временем в число типа UInt8, содержащее номер квартала.

Синоним: QUARTER.

###### toMonth

Переводит дату или дату-с-временем в число типа UInt8, содержащее номер месяца (1-12).

Синоним: MONTH.

###### toDayOfYear

Переводит дату или дату-с-временем в число типа UInt16, содержащее номер дня года (1-366).

Синоним: DAYOFYEAR.

###### toDayOfMonth

Переводит дату или дату-с-временем в число типа UInt8, содержащее номер дня в месяце (1-31).

Синонимы: DAYOFMONTH, DAY.

###### toDayOfWeek

Переводит дату или дату-с-временем в число типа UInt8, содержащее номер дня в неделе (понедельник – 1, воскресенье – 7).

Синоним: DAYOFWEEK.

###### toHour

Переводит дату-с-временем в число типа UInt8, содержащее номер часа в сутках (0-23).

Функция исходит из допущения, что перевод стрелок вперёд, если осуществляется, то на час, в два часа ночи, а перевод стрелок назад, если осуществляется, то на час, в три часа ночи (что, в общем, не верно – даже в Москве два раза перевод стрелок был осуществлён в другое время).

Синоним: HOUR.

###### toMinute

Переводит дату-с-временем в число типа UInt8, содержащее номер минуты в часе (0-59).

Синоним: MINUTE.

###### toSecond

Переводит дату-с-временем в число типа UInt8, содержащее номер секунды в минуте (0-59).

Секунды координации не учитываются.

Синоним: SECOND.

###### toUnixTimestamp

Переводит дату-с-временем в число типа UInt32 -- Unix Timestamp (<https://en.wikipedia.org/wiki/Unix_time>).

Для аргумента String, строка конвертируется в дату и время в соответствии с часовым поясом (необязательный второй аргумент, часовой пояс сервера используется по умолчанию).

Синтаксис:

toUnixTimestamp(datetime)

toUnixTimestamp(str, [timezone])

Возвращаемое значение:

* Возвращает Unix Timestamp.

Тип: UInt32.

Пример:

Запрос:

**SELECT** toUnixTimestamp('2017-11-05 08:07:47', 'Asia/Tokyo') **AS** unix\_timestamp

Результат:

┌─unix\_timestamp─┐

│ 1509836867 │

└────────────────┘

###### toStartOfYear

Округляет дату или дату-с-временем вниз до первого дня года.  
Возвращается дата.

###### toStartOfISOYear

Округляет дату или дату-с-временем вниз до первого дня ISO года. Возвращается дата.

Начало ISO года отличается от начала обычного года, потому что в соответствии с [ISO 8601:1988](https://en.wikipedia.org/wiki/ISO_8601) первая неделя года - это неделя с четырьмя или более днями в этом году.

1 Января 2017 г. – воскресение, т.е. первая ISO неделя 2017 года началась в понедельник 2 января, поэтому 1 января 2017 это 2016 ISO-год, который начался 2016-01-04:

**SELECT** toStartOfISOYear(toDate('2017-01-01')) **AS** ISOYear20170101;

┌─ISOYear20170101─┐

│ 2016-01-04 │

└─────────────────┘

###### toStartOfQuarter

Округляет дату или дату-с-временем вниз до первого дня квартала.

Первый день квартала – это одно из 1 января, 1 апреля, 1 июля, 1 октября.

Возвращается дата.

###### toStartOfMonth

Округляет дату или дату-с-временем вниз до первого дня месяца.

Возвращается дата.

***Внимание:*** *Возвращаемое значение для некорректных дат зависит от реализации. RT.WideStore может вернуть нулевую дату, выбросить исключение, или выполнить «естественное» перетекание дат между месяцами.*

###### toMonday

Округляет дату или дату-с-временем вниз до ближайшего понедельника.  
Возвращается дата.

###### toStartOfWeek(t[,mode])

Округляет дату или дату со временем до ближайшего воскресенья или понедельника в соответствии с mode.

Возвращается дата.

Аргумент mode работает точно так же, как аргумент mode [toWeek()](https://clickhouse.tech/docs/ru/sql-reference/functions/date-time-functions/#toweek). Если аргумент mode опущен, то используется режим 0.

###### toStartOfDay

Округляет дату-с-временем вниз до начала дня. Возвращается дата-с-временем.

###### toStartOfHour

Округляет дату-с-временем вниз до начала часа.

###### toStartOfMinute

Округляет дату-с-временем вниз до начала минуты.

###### toStartOfSecond

Отсекает доли секунды.

Синтаксис:

toStartOfSecond(value[, timezone])

Параметры:

* value – дата и время. [DateTime64](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime64/).
* timezone – [часовой пояс](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/#server_configuration_parameters-timezone) для возвращаемого значения (необязательно). Если параметр не задан, используется часовой пояс параметра value. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).

Возвращаемое значение:

* Входное значение с отсеченными долями секунды.

Тип: [DateTime64](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime64/).

Примеры:

Пример без часового пояса:

**WITH** toDateTime64('2020-01-01 10:20:30.999', 3) **AS** dt64 **SELECT** toStartOfSecond(dt64);

Результат:

┌───toStartOfSecond(dt64)─┐

│ 2020-01-01 10:20:30.000 │

└─────────────────────────┘

Пример с часовым поясом:

**WITH** toDateTime64('2020-01-01 10:20:30.999', 3) **AS** dt64 **SELECT** toStartOfSecond(dt64, 'Europe/Moscow');

Результат:

┌─toStartOfSecond(dt64, 'Europe/Moscow')─┐

│ 2020-01-01 13:20:30.000 │

└────────────────────────────────────────┘

См. также:

* Часовая зона сервера, конфигурационный параметр [timezone](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/#server_configuration_parameters-timezone).

###### toStartOfFiveMinute

Округляет дату-с-временем вниз до начала пятиминутного интервала.

###### toStartOfTenMinutes

Округляет дату-с-временем вниз до начала десятиминутного интервала.

###### toStartOfFifteenMinutes

Округляет дату-с-временем вниз до начала пятнадцатиминутного интервала.

###### toStartOfInterval(time\_or\_data, INTERVAL x unit [, time\_zone])

Обобщение остальных функций toStartOf\*. Например,

toStartOfInterval(t, INTERVAL 1 year) возвращает то же самое, что и toStartOfYear(t),

toStartOfInterval(t, INTERVAL 1 month) возвращает то же самое, что и toStartOfMonth(t),

toStartOfInterval(t, INTERVAL 1 day) возвращает то же самое, что и toStartOfDay(t),

toStartOfInterval(t, INTERVAL 15 minute) возвращает то же самое, что и toStartOfFifteenMinutes(t), и т.п.

###### toTime

Переводит дату-с-временем на некоторую фиксированную дату, сохраняя при этом время.

###### toRelativeYearNum

Переводит дату-с-временем или дату в номер года, начиная с некоторого фиксированного момента в прошлом.

###### toRelativeQuarterNum

Переводит дату-с-временем или дату в номер квартала, начиная с некоторого фиксированного момента в прошлом.

###### toRelativeMonthNum

Переводит дату-с-временем или дату в номер месяца, начиная с некоторого фиксированного момента в прошлом.

###### toRelativeWeekNum

Переводит дату-с-временем или дату в номер недели, начиная с некоторого фиксированного момента в прошлом.

###### toRelativeDayNum

Переводит дату-с-временем или дату в номер дня, начиная с некоторого фиксированного момента в прошлом.

###### toRelativeHourNum

Переводит дату-с-временем в номер часа, начиная с некоторого фиксированного момента в прошлом.

###### toRelativeMinuteNum

Переводит дату-с-временем в номер минуты, начиная с некоторого фиксированного момента в прошлом.

###### toRelativeSecondNum

Переводит дату-с-временем в номер секунды, начиная с некоторого фиксированного момента в прошлом.

###### toISOYear

Переводит дату-с-временем или дату в число типа UInt16, содержащее номер ISO года. ISO год отличается от обычного года, потому что в соответствии с [ISO 8601:1988](https://en.wikipedia.org/wiki/ISO_8601) ISO год начинается необязательно первого января.

Пример:

Запрос:

**SELECT**

toDate('2017-01-01') **AS** date,

toYear(date),

toISOYear(date)

Результат:

┌───────date─┬─toYear(toDate('2017-01-01'))─┬─toISOYear(toDate('2017-01-01'))─┐

│ 2017-01-01 │ 2017 │ 2016 │

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

###### toISOWeek

Переводит дату-с-временем или дату в число типа UInt8, содержащее номер ISO недели.

Начало ISO года отличается от начала обычного года, потому что в соответствии с [ISO 8601:1988](https://en.wikipedia.org/wiki/ISO_8601) первая неделя года – это неделя с четырьмя или более днями в этом году.

1 Января 2017 г. – воскресение, т.е. первая ISO неделя 2017 года началась в понедельник 2 января, поэтому 1 января 2017 это последняя неделя 2016 года.

Пример:

Запрос:

**SELECT**

toISOWeek(toDate('2017-01-01')) **AS** ISOWeek20170101,

toISOWeek(toDate('2017-01-02')) **AS** ISOWeek20170102

Результат:

┌─ISOWeek20170101─┬─ISOWeek20170102─┐

│ 52 │ 1 │

└─────────────────┴─────────────────┘

###### toWeek(date[, mode][, timezone])

Переводит дату-с-временем или дату в число UInt8, содержащее номер недели. Второй аргументам mode задает режим, начинается ли неделя с воскресенья или с понедельника и должно ли возвращаемое значение находиться в диапазоне от 0 до 53 или от 1 до 53. Если аргумент mode опущен, то используется режим 0.

toISOWeek() эквивалентно toWeek(date,3).

Описание режимов (mode):

| ***Mode*** | ***Первый день недели*** | ***Диапазон*** | ***Неделя 1 это первая неделя …*** | |
| --- | --- | --- | --- | --- |
| 0 | Воскресенье | 0-53 | с воскресеньем в этом году | |
| 1 | Понедельник | 0-53 | с 4-мя или более днями в этом году | |
| 2 | Воскресенье | 1-53 | с воскресеньем в этом году | |
| 3 | Понедельник | 1-53 | с 4-мя или более днями в этом году | |
| 4 | Воскресенье | 0-53 | с 4-мя или более днями в этом году | |
| 5 | Понедельник | 0-53 | с понедельником в этом году |
| 6 | Воскресенье | 1-53 | с 4-мя или более днями в этом году |
| 7 | Понедельник | 1-53 | с понедельником в этом году |
| 8 | Воскресенье | 1-53 | содержащая 1 января |
| 9 | Понедельник | 1-53 | содержащая 1 января |

Для режимов со значением «с 4 или более днями в этом году» недели нумеруются в соответствии с ISO 8601:1988:

* Если неделя, содержащая 1 января, имеет 4 или более дней в новом году, это неделя 1.
* В противном случае это последняя неделя предыдущего года, а следующая неделя – неделя 1.

Для режимов со значением «содержит 1 января», неделя 1 – это неделя, содержащая 1 января. Не имеет значения, сколько дней в новом году содержала неделя, даже если она содержала только один день.

Пример:

Запрос:

**SELECT** toDate('2016-12-27') **AS** date, toWeek(date) **AS** week0, toWeek(date,1) **AS** week1, toWeek(date,9) **AS** week9;

Результат:

┌───────date─┬─week0─┬─week1─┬─week9─┐

│ 2016-12-27 │ 52 │ 52 │ 1 │

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

###### toYearWeek(date[,mode])

Возвращает год и неделю для даты. Год в результате может отличаться от года в аргументе даты для первой и последней недели года.

Аргумент mode работает точно так же, как аргумент mode [toWeek()](https://clickhouse.tech/docs/ru/sql-reference/functions/date-time-functions/#toweek). Если mode не задан, используется режим 0.

toISOYear() эквивалентно intDiv(toYearWeek(date,3),100).

Пример:

Запрос:

**SELECT** toDate('2016-12-27') **AS** date, toYearWeek(date) **AS** yearWeek0, toYearWeek(date,1) **AS** yearWeek1, toYearWeek(date,9) **AS** yearWeek9;

Результат:

┌───────date─┬─yearWeek0─┬─yearWeek1─┬─yearWeek9─┐

│ 2016-12-27 │ 201652 │ 201652 │ 201701 │

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

###### date\_trunc

Отсекает от даты и времени части, меньшие чем указанная часть.

Синтаксис:

date\_trunc(unit, value[, timezone])

Синоним: dateTrunc.

Аргументы:

* unit – название части даты или времени. [String Literal](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-string-literal).
* Возможные значения:
  + second,
  + minute,
  + hour,
  + day,
  + week,
  + month,
  + quarter,
  + year.
* value – дата и время. [DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/) или [DateTime64](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime64/).
* timezone – [часовой пояс](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/#server_configuration_parameters-timezone) для возвращаемого значения (необязательно). Если параметр не задан, используется часовой пояс параметра value. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)

Возвращаемое значение:

* Дата и время, отсеченные до указанной части.

Тип: [Datetime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/).

Примеры:

Запрос без указания часового пояса:

**SELECT** now(), date\_trunc('hour', now());

Результат:

┌───────────────now()─┬─date\_trunc('hour', now())─┐

│ 2020-09-28 10:40:45 │ 2020-09-28 10:00:00 │

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

Запрос с указанием часового пояса:

**SELECT** now(), date\_trunc('hour', now(), 'Europe/Moscow');

Результат:

┌───────────────now()─┬─date\_trunc('hour', now(), 'Europe/Moscow')─┐

│ 2020-09-28 10:46:26 │ 2020-09-28 13:00:00 │

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

См. также:

* [toStartOfInterval](https://clickhouse.tech/docs/ru/sql-reference/functions/date-time-functions/#tostartofintervaltime-or-data-interval-x-unit-time-zone).

###### now

Возвращает текущую дату и время.

Синтаксис:

now([timezone])

Параметры

* timezone – [часовой пояс](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/#server_configuration_parameters-timezone) для возвращаемого значения (необязательно). [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).

Возвращаемое значение:

* Текущие дата и время.

Тип: [Datetime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/).

Пример:

Запрос без указания часового пояса:

**SELECT** now();

Результат:

┌───────────────now()─┐

│ 2020-10-17 07:42:09 │

└─────────────────────┘

Запрос с указанием часового пояса:

**SELECT** now('Europe/Moscow');

Результат:

┌─now('Europe/Moscow')─┐

│ 2020-10-17 10:42:23 │

└──────────────────────┘

###### today

Принимает ноль аргументов и возвращает текущую дату на один из моментов выполнения запроса.

То же самое, что toDate(now()).

###### yesterday

Принимает ноль аргументов и возвращает вчерашнюю дату на один из моментов выполнения запроса.

Делает то же самое, что today() - 1.

###### dateDiff

Вычисляет разницу между двумя значениями дат с временем.

Синтаксис:

dateDiff('unit', startdate, enddate, [timezone])

Параметры:

* unit – Единица измерения времени, в которой будет вычислена разница между startdate и enddate. [String](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-string-literal).

Поддерживаемые значения:

| unit |

| ------ |

|**second** |

|**minute** |

|hour |

|**day** |

|week |

|**month** |

|quarter |

|**year** |

* startdate – первая дата. [Date](https://clickhouse.tech/docs/ru/sql-reference/functions/date-time-functions/) или [DateTime](https://clickhouse.tech/docs/ru/sql-reference/functions/date-time-functions/).
* enddate – вторая дата. [Date](https://clickhouse.tech/docs/ru/sql-reference/functions/date-time-functions/) или [DateTime](https://clickhouse.tech/docs/ru/sql-reference/functions/date-time-functions/).
* timezone – опциональный параметр. Если определен, применяется к обоим значениям: startdate и enddate. Если не определен, используются часовые пояса startdate и enddate. Если часовые пояса не совпадают, вернется неожидаемый результат.

Возвращаемое значение:

Разница между startdate и enddate, выраженная в unit.

Тип: int.

Пример:

Запрос:

**SELECT** dateDiff('hour', toDateTime('2018-01-01 22:00:00'), toDateTime('2018-01-02 23:00:00'));

Результат:

┌─dateDiff('hour', toDateTime('2018-01-01 22:00:00'), toDateTime('2018-01-02 23:00:00'))─┐

│ 25 │

└────────────────────────────────────────────────────────────────────────────────────────┘

###### date\_sub

Вычитает интервал времени или даты из указанной даты или даты со временем.

Синтаксис:

date\_sub(unit, value, date)

Синонимы: dateSub, DATE\_SUB.

Аргументы:

* unit – единица измерения времени, в которой задан интервал для вычитания. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/). Возможные значения:
  + second,
  + minute,
  + hour,
  + day,
  + week,
  + month,
  + quarter,
  + year.
* value – значение интервала для вычитания. [Int](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).
* date – дата или дата со временем, из которой вычитается value. [Date](https://clickhouse.tech/docs/ru/sql-reference/data-types/date/) или [DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/).

Возвращаемое значение:

* Возвращает дату или дату со временем, полученную в результате вычитания value, выраженного в unit, из date.

Тип: [Date](https://clickhouse.tech/docs/ru/sql-reference/data-types/date/) или [DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/).

Пример:

Запрос:

**SELECT** date\_sub(**YEAR**, 3, toDate('2018-01-01'));

Результат:

┌─minus(toDate('2018-01-01'), toIntervalYear(3))─┐

│ 2015-01-01 │

└────────────────────────────────────────────────┘

###### timeSlot

Округляет время до получаса.

Эта функция является специфичной, так как пол часа - минимальное время, для которого, если соседние по времени хиты одного посетителя на одном счётчике отстоят друг от друга строго более, чем на это время, визит может быть разбит на два визита. То есть, кортежи (номер счётчика, идентификатор посетителя, тайм-слот) могут использоваться для поиска хитов, входящий в соответствующий визит.

###### timeSlots(StartTime, Duration,[, Size])

Для интервала времени, начинающегося в ‘StartTime’ и продолжающегося ‘Duration’ секунд, возвращает массив моментов времени, состоящий из округлений вниз до ‘Size’ точек в секундах из этого интервала. ‘Size’ - необязательный параметр, константный UInt32, по умолчанию равен 1800.

Например, timeSlots(toDateTime('2012-01-01 12:20:00'), toUInt32(600)) = [toDateTime('2012-01-01 12:00:00'), toDateTime('2012-01-01 12:30:00')].

Это нужно для поиска хитов, входящих в соответствующий визит.

###### toYYYYMM

Переводит дату или дату со временем в число типа UInt32, содержащее номер года и месяца (YYYY \* 100 + MM).

###### toYYYYMMDD

Переводит дату или дату со временем в число типа UInt32, содержащее номер года, месяца и дня (YYYY \* 10000 + MM \* 100 + DD).

###### toYYYYMMDDhhmmss

Переводит дату или дату со временем в число типа UInt64 содержащее номер года, месяца, дня и время (YYYY \* 10000000000 + MM \* 100000000 + DD \* 1000000 + hh \* 10000 + mm \* 100 + ss).

###### formatDateTime

Функция преобразует дату-и-время в строку по заданному шаблону. Важно: шаблон – константное выражение, поэтому использовать разные шаблоны в одной колонке не получится.

Синтаксис:

formatDateTime(Time, Format\[, Timezone\])

Возвращаемое значение:

* Возвращает значение времени и даты в определенном вами формате.

Поля подстановки:

Используйте поля подстановки для того, чтобы определить шаблон для выводимой строки. В колонке «Пример» результат работы функции для времени 2018-01-02 22:33:44.

| ***Поле*** | ***Описание*** | ***Пример*** |
| --- | --- | --- |
| %C | номер года, поделённый на 100 (00-99) | 20 |
| %d | день месяца, с ведущим нулём (01-31) | 02 |
| %D | короткая запись %m/%d/%y | 01/02/18 |
| %e | день месяца, с ведущим пробелом ( 1-31) | 2 |
| %F | короткая запись %Y-%m-%d | 2018-01-02 |
| %G | четырехзначный формат вывода ISO-года, который основывается на особом подсчете номера недели согласно [стандарту ISO 8601](https://ru.wikipedia.org/wiki/ISO_8601), обычно используется вместе с %V | 2018 |
| %g | двузначный формат вывода года по стандарту ISO 8601 | 18 |
| %H | час в 24-часовом формате (00-23) | 22 |
| %I | час в 12-часовом формате (01-12) | 10 |
| %j | номер дня в году, с ведущими нулями (001-366) | 002 |
| %m | месяц, с ведущим нулём (01-12) | 01 |
| %M | минуты, с ведущим нулём (00-59) | 33 |
| %n | символ переноса строки (‘’) |  |
| %p | обозначения AM или PM | PM |
| %R | короткая запись %H:%M | 22:33 |
| %S | секунды, с ведущими нулями (00-59) | 44 |
| %t | символ табуляции (’) |  |
| %T | формат времени ISO 8601, одинаковый с %H:%M:%S | 22:33:44 |
| %u | номер дня недели согласно ISO 8601, понедельник - 1, воскресенье - 7 | 2 |
| %V | номер недели согласно ISO 8601 (01-53) | 01 |
| %w | номер дня недели, начиная с воскресенья (0-6) | 2 |
| %y | год, последние 2 цифры (00-99) | 18 |
| %Y | год, 4 цифры | 2018 |
| %% | символ % | % |

Пример:

Запрос:

**SELECT** formatDateTime(toDate('2010-01-04'), '%g');

Результат:

┌─formatDateTime(toDate('2010-01-04'), '%g')─┐

│ 10 │

└────────────────────────────────────────────┘

###### FROM\_UNIXTIME

Функция преобразует Unix timestamp в календарную дату и время.

Примеры:

Если указан только один аргумент типа [Integer](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/), то функция действует так же, как [toDateTime](https://clickhouse.tech/docs/ru/sql-reference/functions/type-conversion-functions/#todatetime), и возвращает тип [DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/).

Запрос:

**SELECT** FROM\_UNIXTIME(423543535);

Результат:

┌─FROM\_UNIXTIME(423543535)─┐

│ 1983-06-04 10:58:55 │

└──────────────────────────┘

В случае, когда есть два аргумента: первый типа [Integer](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/) или [DateTime](https://clickhouse.tech/docs/ru/sql-reference/data-types/datetime/), а второй является строкой постоянного формата – функция работает также, как [formatDateTime](https://clickhouse.tech/docs/ru/sql-reference/functions/date-time-functions/#formatdatetime), и возвращает значение типа [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/#string).

Запрос:

**SELECT** FROM\_UNIXTIME(1234334543, '%Y-%m-%d %R:%S') **AS** DateTime;

Результат:

┌─DateTime────────────┐

│ 2009-02-11 14:42:23 │

└─────────────────────┘

##### Функции для работы со строками

###### empty

Возвращает 1 для пустой строки, и 0 для непустой строки.

Тип результата – UInt8.

Строка считается непустой, если содержит хотя бы один байт, пусть даже это пробел или нулевой байт.

Функция также работает для массивов.

###### notEmpty

Возвращает 0 для пустой строки, и 1 для непустой строки.

Тип результата – UInt8.

Функция также работает для массивов.

###### length

Возвращает длину строки в байтах (не символах, не кодовых точках).

Тип результата – UInt64.

Функция также работает для массивов.

###### lengthUTF8

Возвращает длину строки в кодовых точках Unicode (не символах), при допущении, что строка содержит набор байтов, являющийся текстом в кодировке UTF-8. Если допущение не выполнено, то возвращает какой-нибудь результат (не кидает исключение).

Тип результата – UInt64.

###### char\_length, CHAR\_LENGTH

Возвращает длину строки в кодовых точках Unicode (не символах), при допущении, что строка содержит набор байтов, являющийся текстом в кодировке UTF-8. Если допущение не выполнено, возвращает какой-нибудь результат (не кидает исключение).

Тип результата – UInt64.

###### character\_length, CHARACTER\_LENGTH

Возвращает длину строки в кодовых точках Unicode (не символах), при допущении, что строка содержит набор байтов, являющийся текстом в кодировке UTF-8. Если допущение не выполнено, возвращает какой-нибудь результат (не кидает исключение).

Тип результата – UInt64.

###### lower, lcase

Переводит ASCII-символы латиницы в строке в нижний регистр.

###### upper, ucase

Переводит ASCII-символы латиницы в строке в верхний регистр.

###### lowerUTF8

Переводит строку в нижний регистр, при допущении, что строка содержит набор байтов, представляющий текст в кодировке UTF-8.

Не учитывает язык. То есть, для турецкого языка, результат может быть не совсем верным.

Если длина UTF-8 последовательности байтов различна для верхнего и нижнего регистра кодовой точки, то для этой кодовой точки результат работы может быть некорректным.

Если строка содержит набор байтов, не являющийся UTF-8, то поведение не определено.

###### upperUTF8

Переводит строку в верхний регистр, при допущении, что строка содержит набор байтов, представляющий текст в кодировке UTF-8.

Не учитывает язык. То есть, для турецкого языка, результат может быть не совсем верным.

Если длина UTF-8 последовательности байтов различна для верхнего и нижнего регистра кодовой точки, то для этой кодовой точки, результат работы может быть некорректным.

Если строка содержит набор байтов, не являющийся UTF-8, то поведение не определено.

###### isValidUTF8

Возвращает 1, если набор байтов является корректным в кодировке UTF-8, 0 иначе.

###### toValidUTF8

Заменяет некорректные символы UTF-8 на символ � (U+FFFD). Все идущие подряд некорректные символы схлопываются в один заменяющий символ.

Синтаксис:

toValidUTF8( input\_string )

Параметры:

* input\_string – произвольный набор байтов, представленный как объект типа [String](https://clickhouse.tech/docs/ru/sql-reference/functions/string-functions/).

Возвращаемое значение: Корректная строка UTF-8.

Пример:

**SELECT** toValidUTF8('\x61\xF0\x80\x80\x80b')

┌─toValidUTF8('a����b')─┐

│ a�b │

└───────────────────────┘

###### repeat

Повторяет строку определенное количество раз и объединяет повторяемые значения в одну строку.

Синоним: REPEAT.

Синтаксис:

repeat(s, n)

Параметры:

* s – строка для повторения. [String](https://clickhouse.tech/docs/ru/sql-reference/functions/string-functions/).
* n – количество повторов. [UInt](https://clickhouse.tech/docs/ru/sql-reference/functions/string-functions/).

Возвращаемое значение:

* Строка, состоящая из повторений n раз исходной строки s. Если n \< 1, то функция вернет пустую строку.

Тип: String.

Пример:

Запрос:

**SELECT** repeat('abc', 10)

Ответ:

┌─repeat('abc', 10)──────────────┐

│ abcabcabcabcabcabcabcabcabcabc │

└────────────────────────────────┘

###### reverse

Разворачивает строку (как последовательность байтов).

###### reverseUTF8

Разворачивает последовательность кодовых точек Unicode, при допущении, что строка содержит набор байтов, представляющий текст в кодировке UTF-8. Иначе – что-то делает (не кидает исключение).

###### format(pattern, s0, s1, …)

Форматирует константный шаблон со строками, перечисленными в аргументах. pattern – упрощенная версия шаблона в языке Python. Шаблон содержит «заменяющие поля», которые окружены фигурными скобками {}. Всё, что не содержится в скобках, интерпретируется как обычный текст и просто копируется. Если нужно использовать символ фигурной скобки, можно экранировать двойной скобкой {{ или }}. Имя полей могут быть числами (нумерация с нуля) или пустыми (тогда они интерпретируются как последовательные числа).

**SELECT** format('{1} {0} {1}', 'World', 'Hello')

┌─format('{1} {0} {1}', 'World', 'Hello')─┐

│ Hello World Hello │

└─────────────────────────────────────────┘

**SELECT** format('{} {}', 'Hello', 'World')

┌─format('{} {}', 'Hello', 'World')─┐

│ Hello World │

└───────────────────────────────────┘

###### concat

Склеивает строки, переданные в аргументы, в одну строку без разделителей.

Cинтаксис:

concat(s1, s2, ...)

Параметры:

* Значения типа String или FixedString.

Возвращаемое значение:

* Возвращает строку, полученную в результате склейки аргументов.

Если любой из аргументов имеет значение NULL, concat возвращает значение NULL.

Пример:

Запрос:

**SELECT** concat('Hello, ', 'World!')

Ответ:

┌─concat('Hello, ', 'World!')─┐

│ Hello, World! │

└─────────────────────────────┘

###### concatAssumeInjective

Аналогична [concat](https://clickhouse.tech/docs/ru/sql-reference/functions/string-functions/#concat). Разница заключается в том, что вам нужно убедиться, что concat(s1, s2, ...) → sn является инъективным, так как это предположение будет использоваться для оптимизации GROUP BY.

Функция называется «инъективной», если она возвращает разные значения для разных аргументов. Или, иными словами, функция никогда не выдаёт одно и то же значение, если аргументы разные.

Синтаксис:

concatAssumeInjective(s1, s2, ...)

Параметры:

* Значения типа String или FixedString.

Возвращаемые значения:

* Возвращает строку, полученную в результате объединения аргументов.

Если любой из аргументов имеет значение NULL, concatAssumeInjective возвращает значение NULL.

Пример:

Вводная таблица:

**CREATE** **TABLE** key\_val(`key1` String, `key2` String, `value` UInt32) ENGINE = TinyLog

**INSERT** **INTO** key\_val **VALUES** ('Hello, ','World',1)('Hello, ','World',2)('Hello, ','World!',3)('Hello',', World!',2)

**SELECT** \* **from** key\_val

┌─key1────┬─key2─────┬─value─┐

│ Hello, │ World │ 1 │

│ Hello, │ World │ 2 │

│ Hello, │ World! │ 3 │

│ Hello │ , World! │ 2 │

└─────────┴──────────┴───────┘

Запрос:

**SELECT** concat(key1, key2), **sum**(value) **FROM** key\_val **GROUP** **BY** (key1, key2)

Ответ:

┌─concat(key1, key2)─┬─sum(value)─┐

│ Hello, World! │ 3 │

│ Hello, World! │ 2 │

│ Hello, World │ 3 │

└────────────────────┴────────────┘

###### substring(s, offset, length), mid(s, offset, length), substr(s, offset, length)

Возвращает подстроку, начиная с байта по индексу offset, длины length байт. Индексация символов – начиная с единицы (как в стандартном SQL). Аргументы offset и length должны быть константами.

###### substringUTF8(s, offset, length)

Так же, как substring, но для кодовых точек Unicode. Работает при допущении, что строка содержит набор байтов, представляющий текст в кодировке UTF-8. Если допущение не выполнено, то возвращает какой-нибудь результат (не кидает исключение).

###### appendTrailingCharIfAbsent(s, c)

Если строка s непустая и не содержит символ c на конце, то добавляет символ c в конец.

###### convertCharset(s, from, to)

Возвращает сконвертированную из кодировки from в кодировку to строку s.

###### base64Encode(s)

Производит кодирование строки s в base64-представление.

Синоним: TO\_BASE64.

###### base64Decode(s)

Декодирует base64-представление s в исходную строку. При невозможности декодирования выбрасывает исключение

Синоним: FROM\_BASE64.

###### tryBase64Decode(s)

Функционал аналогичен base64Decode, но при невозможности декодирования возвращает пустую строку.

###### endsWith(s, suffix)

Возвращает 1, если строка завершается указанным суффиксом, и 0 в противном случае.

###### startsWith(str, prefix)

Возвращает 1, если строка начинается указанным префиксом, в противном случае 0.

**SELECT** startsWith('Spider-Man', 'Spi');

Возвращаемые значения:

* 1, если строка начинается указанным префиксом.
* 0, если строка не начинается указанным префиксом.

Пример:

Запрос:

**SELECT** startsWith('Hello, world!', 'He');

Ответ:

┌─startsWith('Hello, world!', 'He')─┐

│ 1 │

└───────────────────────────────────┘

###### trim

Удаляет все указанные символы с начала или окончания строки.

По умолчанию удаляет все последовательные вхождения обычных пробелов (32 символ ASCII) с обоих концов строки.

Синтаксис:

**trim**([[**LEADING**|**TRAILING**|**BOTH**] trim\_character **FROM**] input\_string)

Параметры:

* trim\_character – один или несколько символов, подлежащие удалению. [String](https://clickhouse.tech/docs/ru/sql-reference/functions/string-functions/).
* input\_string – строка для обрезки. [String](https://clickhouse.tech/docs/ru/sql-reference/functions/string-functions/).

Возвращаемое значение:

* Исходную строку после обрезки с левого и (или) правого концов строки.

Тип: String.

Пример:

Запрос:

**SELECT** **trim**(**BOTH** ' ()' **FROM** '( Hello, world! )')

Ответ:

┌─trim(BOTH ' ()' FROM '( Hello, world! )')─┐

│ Hello, world! │

└───────────────────────────────────────────────┘

###### trimLeft

Удаляет все последовательные вхождения обычных пробелов (32 символ ASCII) с левого конца строки. Не удаляет другие виды пробелов (табуляция, пробел без разрыва и т. д.).

Синтаксис:

trimLeft(input\_string)

Алиас: ltrim(input\_string).

Параметры:

* input\_string – строка для обрезки. [String](https://clickhouse.tech/docs/ru/sql-reference/functions/string-functions/).

Возвращаемое значение:

* Исходную строку без общих пробельных символов слева.

Тип: String.

Пример:

Запрос:

**SELECT** trimLeft(' Hello, world! ')

Ответ:

┌─trimLeft(' Hello, world! ')─┐

│ Hello, world! │

└─────────────────────────────────────┘

###### trimRight

Удаляет все последовательные вхождения обычных пробелов (32 символ ASCII) с правого конца строки. Не удаляет другие виды пробелов (табуляция, пробел без разрыва и т. д.).

Синтаксис:

trimRight(input\_string)

Алиас: rtrim(input\_string).

Параметры:

* input\_string – строка для обрезки. [String](https://clickhouse.tech/docs/ru/sql-reference/functions/string-functions/).

Возвращаемое значение:

* Исходную строку без общих пробельных символов справа.

Тип: String.

Пример:

Запрос:

**SELECT** trimRight(' Hello, world! ')

Ответ:

┌─trimRight(' Hello, world! ')─┐

│ Hello, world! │

└──────────────────────────────────────┘

###### trimBoth

Удаляет все последовательные вхождения обычных пробелов (32 символ ASCII) с обоих концов строки. Не удаляет другие виды пробелов (табуляция, пробел без разрыва и т. д.).

Синтаксис:

trimBoth(input\_string)

Алиас: trim(input\_string).

Параметры:

* input\_string – строка для обрезки. [String](https://clickhouse.tech/docs/ru/sql-reference/functions/string-functions/).

Возвращаемое значение:

* Исходную строку без общих пробельных символов с обоих концов строки.

Тип: String.

Пример:

Запрос:

**SELECT** trimBoth(' Hello, world! ')

Ответ:

┌─trimBoth(' Hello, world! ')─┐

│ Hello, world! │

└─────────────────────────────────────┘

###### CRC32(s)

Возвращает чексумму CRC32 данной строки, используется CRC-32-IEEE 802.3 многочлен и начальным значением 0xffffffff (т.к. используется реализация из zlib).

Тип результата – UInt32.

###### CRC32IEEE(s)

Возвращает чексумму CRC32 данной строки, используется CRC-32-IEEE 802.3 многочлен.

Тип результата – UInt32.

###### CRC64(s)

Возвращает чексумму CRC64 данной строки, используется CRC-64-ECMA многочлен.

Тип результата – UInt64.

###### normalizeQuery

Заменяет литералы, последовательности литералов и сложные псевдонимы заполнителями.

Синтаксис:

normalizeQuery(x)

Параметры:

* x – последовательность символов. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).

Возвращаемое значение:

* Последовательность символов с заполнителями.

Тип: [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).

Пример:

Запрос:

**SELECT** normalizeQuery('[1, 2, 3, x]') **AS** query;

Результат:

┌─query────┐

│ [?.., x] │

└──────────┘

###### normalizedQueryHash

Возвращает идентичные 64-битные хэш - суммы без значений литералов для аналогичных запросов. Это помогает анализировать журнал запросов.

Синтаксис:

normalizedQueryHash(x)

Параметры:

* x – последовательность символов. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).

Возвращаемое значение:

* Хэш-сумма.

Тип: [UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges).

Пример:

Запрос:

**SELECT** normalizedQueryHash('SELECT 1 AS `xyz`') != normalizedQueryHash('SELECT 1 AS `abc`') **AS** res;

Результат:

┌─res─┐

│ 1 │

└─────┘

###### encodeXMLComponent

Экранирует символы для размещения строки в текстовом узле или атрибуте XML.

Экранируются символы, которые в формате XML являются зарезервированными (служебными): <, &, >, ", '.

Синтаксис:

encodeXMLComponent(x)

Параметры:

* x – последовательность символов. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).

Возвращаемое значение:

* Строка, в которой зарезервированные символы экранированы.

Тип: [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).

Пример:

Запрос:

**SELECT** encodeXMLComponent('Hello, "world"!');

**SELECT** encodeXMLComponent('<123>');

**SELECT** encodeXMLComponent('&clickhouse');

**SELECT** encodeXMLComponent('\'foo\'');

Результат:

Hello, &quot;world&quot;!

&lt;123&gt;

&amp;clickhouse

&apos;foo&apos;

###### decodeXMLComponent

Заменяет символами предопределенные мнемоники XML: &quot; &amp; &apos; &gt; &lt.

Также эта функция заменяет числовые ссылки соответствующими символами юникод. Поддерживаются десятичная (например, &#10003;) и шестнадцатеричная (&#x2713;) формы.

Синтаксис:

decodeXMLComponent(x)

Параметры:

* x – последовательность символов. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).

Возвращаемое значение:

* Строка с произведенными заменами.

Тип: [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).

Пример:

Запрос:

**SELECT** decodeXMLComponent('&apos;foo&apos;');

**SELECT** decodeXMLComponent('&lt; &#x3A3; &gt;');

Результат:

'foo'

< Σ >

Смотрите также:

* [Мнемоники в HTML](https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%B5%D0%BC%D0%BE%D0%BD%D0%B8%D0%BA%D0%B8_%D0%B2_HTML).

##### Функции поиска в строках

Во всех функциях, поиск регистрозависимый по умолчанию. Существуют варианты функций для регистронезависимого поиска.

###### position(haystack, needle)

Поиск подстроки needle в строке haystack.

Возвращает позицию (в байтах) найденной подстроки в строке, начиная с 1, или 0, если подстрока не найдена.

Для поиска без учета регистра используйте функцию [positionCaseInsensitive](https://clickhouse.tech/docs/ru/sql-reference/functions/string-search-functions/#positioncaseinsensitive).

Синтаксис:

**position**(haystack, needle[, start\_pos])

Алиас: locate(haystack, needle[, start\_pos]).

Параметры:

* haystack – строка, по которой выполняется поиск. [Строка](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-string-literal).
* needle – подстрока, которую необходимо найти. [Строка](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-string-literal).
* start\_pos – опциональный параметр, позиция символа в строке, с которого начинается поиск. [UInt](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)

Возвращаемые значения:

* Начальная позиция в байтах (начиная с 1), если подстрока найдена.
* 0, если подстрока не найдена.

Тип: Integer.

Примеры:

Фраза «Hello, world!» содержит набор байт, представляющий текст в однобайтовой кодировке. Функция возвращает ожидаемый результат:

Запрос:

**SELECT** **position**('Hello, world!', '!')

Ответ:

┌─position('Hello, world!', '!')─┐

│ 13 │

└────────────────────────────────┘

Аналогичная фраза на русском содержит символы, которые не могут быть представлены в однобайтовой кодировке. Функция возвращает неожиданный результат (используйте функцию [positionUTF8](https://clickhouse.tech/docs/ru/sql-reference/functions/string-search-functions/#positionutf8) для символов, которые не могут быть представлены одним байтом):

Запрос:

**SELECT** **position**('Привет, мир!', '!')

Ответ:

┌─position('Привет, мир!', '!')─┐

│ 21 │

└───────────────────────────────┘

###### positionCaseInsensitive

Такая же, как и [position](https://clickhouse.tech/docs/ru/sql-reference/functions/string-search-functions/#position), но работает без учета регистра. Возвращает позицию в байтах найденной подстроки в строке, начиная с 1.

Работает при допущении, что строка содержит набор байт, представляющий текст в однобайтовой кодировке. Если допущение не выполнено – то возвращает неопределенный результат (не кидает исключение). Если символ может быть представлен с помощью двух байтов, он будет представлен двумя байтами и так далее.

Синтаксис:

positionCaseInsensitive(haystack, needle[, start\_pos])

Параметры:

* haystack – строка, по которой выполняется поиск. [Строка](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-string-literal).
* needle – подстрока, которую необходимо найти. [Строка](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-string-literal).
* start\_pos – опциональный параметр, позиция символа в строке, с которого начинается поиск. [UInt](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)

Возвращаемые значения:

* Начальная позиция в байтах (начиная с 1), если подстрока найдена.
* 0, если подстрока не найдена.

Тип: Integer.

Пример:

Запрос:

**SELECT** positionCaseInsensitive('Hello, world!', 'hello')

Ответ:

┌─positionCaseInsensitive('Hello, world!', 'hello')─┐

│ 1 │

└───────────────────────────────────────────────────┘

###### positionUTF8

Возвращает позицию (в кодовых точках Unicode) найденной подстроки в строке, начиная с 1.

Работает при допущении, что строка содержит набор кодовых точек, представляющий текст в кодировке UTF-8. Если допущение не выполнено – то возвращает неопределенный результат (не кидает исключение). Если символ может быть представлен с помощью двух кодовых точек, он будет представлен двумя и так далее.

Для поиска без учета регистра используйте функцию [positionCaseInsensitiveUTF8](https://clickhouse.tech/docs/ru/sql-reference/functions/string-search-functions/#positioncaseinsensitiveutf8).

Синтаксис:

positionUTF8(haystack, needle[, start\_pos])

Параметры:

* haystack – строка, по которой выполняется поиск. [Строка](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-string-literal).
* needle – подстрока, которую необходимо найти. [Строка](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-string-literal).
* start\_pos – опциональный параметр, позиция символа в строке, с которого начинается поиск. [UInt](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Возвращаемые значения:

* Начальная позиция в кодовых точках Unicode (начиная с 1), если подстрока найдена.
* 0, если подстрока не найдена.

Тип: Integer.

Примеры:

Фраза «Привет, мир!» содержит набор символов, каждый из которых можно представить с помощью одной кодовой точки. Функция возвращает ожидаемый результат:

Запрос:

**SELECT** positionUTF8('Привет, мир!', '!')

Ответ:

┌─positionUTF8('Привет, мир!', '!')─┐

│ 12 │

└───────────────────────────────────┘

Фраза «Salut, étudiante!» содержит символ é, который может быть представлен одной кодовой точкой (U+00E9) или двумя (U+0065U+0301). Поэтому функция positionUTF8() может вернуть неожиданный результат:

Запрос для символа é, который представлен одной кодовой точкой U+00E9:

**SELECT** positionUTF8('Salut, étudiante!', '!')

Ответ:

┌─positionUTF8('Salut, étudiante!', '!')─┐

│ 17 │

└────────────────────────────────────────┘

Запрос для символа é, который представлен двумя кодовыми точками U+0065U+0301:

**SELECT** positionUTF8('Salut, étudiante!', '!')

Ответ:

┌─positionUTF8('Salut, étudiante!', '!')─┐

│ 18 │

└────────────────────────────────────────┘

###### positionCaseInsensitiveUTF8

Такая же, как и [positionUTF8](https://clickhouse.tech/docs/ru/sql-reference/functions/string-search-functions/#positionutf8), но работает без учета регистра. Возвращает позицию (в кодовых точках Unicode) найденной подстроки в строке, начиная с 1.

Работает при допущении, что строка содержит набор кодовых точек, представляющий текст в кодировке UTF-8. Если допущение не выполнено – то возвращает неопределенный результат (не кидает исключение). Если символ может быть представлен с помощью двух кодовых точек, он будет представлен двумя и так далее.

Синтаксис:

positionCaseInsensitiveUTF8(haystack, needle[, start\_pos])

Параметры:

* haystack – строка, по которой выполняется поиск. [Строка](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-string-literal).
* needle – подстрока, которую необходимо найти. [Строка](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-string-literal).
* start\_pos – опциональный параметр, позиция символа в строке, с которого начинается поиск. [UInt](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Возвращаемые значения:

* Начальная позиция в байтах (начиная с 1), если подстрока найдена.
* 0, если подстрока не найдена.

Тип: Integer.

Пример:

Запрос:

**SELECT** positionCaseInsensitiveUTF8('Привет, мир!', 'Мир')

Ответ:

┌─positionCaseInsensitiveUTF8('Привет, мир!', 'Мир')─┐

│ 9 │

└────────────────────────────────────────────────────┘

###### multiSearchAllPositions

То же самое, что и position, но возвращает массив позиций (в байтах) найденных соответствующих подстрок в строке. Позиции индексируются, начиная с 1.

Поиск выполняется по последовательностям байтов без учета строкового кодирования и сортировки.

* Для поиска ASCII без учета регистра используйте функцию multiSearchAllPositions CaseInsensitive.
* Для поиска в UTF-8 используйте функцию multiSearchAllPositionsUTF8.
* Для поиска без учета регистра UTF-8 используйте функцию multi Search All Positions Case Insensitive UTF 8.

Синтаксис:

multiSearchAllPositions(haystack, [needle1, needle2, ..., needlen])

Параметры:

* haystack – строка, в которой будет производиться поиск подстроки. [String](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-string-literal).
* needle – подстрока для поиска. [String](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-string-literal).

Возвращаемые значения:

* Массив начальных позиций в байтах (отсчет от 1), если соответствующая подстрока была найдена, и 0, если не найдена.

Пример:

Запрос:

**SELECT** multiSearchAllPositions('Hello, World!', ['hello', '!', 'world'])

Результат:

┌─multiSearchAllPositions('Hello, World!', ['hello', '!', 'world'])─┐

│ [0,13,0] │

└───────────────────────────────────────────────────────────────────┘

###### multiSearchAllPositionsUTF8

Смотрите multiSearchAllPositions.

###### multiSearchFirstPosition(haystack, [needle1, needle2, …, needlen])

Так же, как и position, только возвращает оффсет первого вхождения любого из needles.

Для поиска без учета регистра и/или в кодировке UTF-8 используйте функции multiSearchFirstPositionCaseInsensitive, multiSearchFirstPositionUTF8, multiSearchFirstPositionCaseInsensitiveUTF8.

###### multiSearchFirstIndex(haystack, [needle1, needle2, …, needlen])

Возвращает индекс i (нумерация с единицы) первой найденной строки needlei в строке haystack и 0 иначе.

Для поиска без учета регистра и/или в кодировке UTF-8 используйте функции multiSearchFirstIndexCaseInsensitive, multiSearchFirstIndexUTF8, multiSearchFirstIndexCaseInsensitiveUTF8.

###### multiSearchAny(haystack, [needle1, needle2, …, needlen])

Возвращает 1, если хотя бы одна подстрока needlei нашлась в строке haystack и 0 иначе.

Для поиска без учета регистра и/или в кодировке UTF-8 используйте функции multiSearchAnyCaseInsensitive, multiSearchAnyUTF8, multiSearchAnyCaseInsensitiveUTF8.

***Примечание:*** *Во всех функциях multiSearch\* количество needles должно быть меньше 28 из-за особенностей реализации.*

###### match(haystack, pattern)

Проверка строки на соответствие регулярному выражению pattern. Регулярное выражение **re2**. Синтаксис регулярных выражений **re2** является более ограниченным по сравнению с регулярными выражениями **Perl** ([подробнее](https://github.com/google/re2/wiki/Syntax)).

Возвращает 0 (если не соответствует) или 1 (если соответствует).

Обратите внимание, что для экранирования в регулярном выражении, используется символ \ (обратный слеш). Этот же символ используется для экранирования в строковых литералах. Поэтому, чтобы экранировать символ в регулярном выражении, необходимо написать в строковом литерале \ (два обратных слеша).

Регулярное выражение работает со строкой как с набором байт. Регулярное выражение не может содержать нулевые байты.

Для шаблонов на поиск подстроки в строке, лучше используйте LIKE или position, так как они работают существенно быстрее.

###### multiMatchAny(haystack, [pattern1, pattern2, …, patternn])

То же, что и match, но возвращает ноль, если ни одно регулярное выражение не подошло и один, если хотя бы одно. Используется библиотека [hyperscan](https://github.com/intel/hyperscan) для соответствия регулярных выражений. Для шаблонов на поиск многих подстрок в строке, лучше используйте multiSearchAny, так как она работает существенно быстрее.

***Примечание:*** *Длина любой строки из haystack должна быть меньше 232 байт, иначе бросается исключение. Это ограничение связано с ограничением hyperscan API.*

###### multiMatchAnyIndex(haystack, [pattern1, pattern2, …, patternn])

То же, что и multiMatchAny, только возвращает любой индекс подходящего регулярного выражения.

###### multiMatchAllIndices(haystack, [pattern1, pattern2, …, patternn])

То же, что и multiMatchAny, только возвращает массив всех индексов всех подходящих регулярных выражений в любом порядке.

###### multiFuzzyMatchAny(haystack, distance, [pattern1, pattern2, …, patternn])

То же, что и multiMatchAny, но возвращает 1 если любой pattern соответствует haystack в пределах константного [редакционного расстояния](https://en.wikipedia.org/wiki/Edit_distance). Эта функция также находится в экспериментальном режиме и может быть очень медленной. За подробностями обращайтесь к [документации hyperscan](https://intel.github.io/hyperscan/dev-reference/compilation.html#approximate-matching).

###### multiFuzzyMatchAnyIndex(haystack, distance, [pattern1, pattern2, …, patternn])

То же, что и multiFuzzyMatchAny, только возвращает любой индекс подходящего регулярного выражения в пределах константного редакционного расстояния.

###### multiFuzzyMatchAllIndices(haystack, distance, [pattern1, pattern2, …, patternn])

То же, что и multiFuzzyMatchAny, только возвращает массив всех индексов всех подходящих регулярных выражений в любом порядке в пределах константного редакционного расстояния.

Примечание

multiFuzzyMatch\* функции не поддерживают UTF-8 закодированные регулярные выражения, и такие выражения рассматриваются как байтовые из-за ограничения hyperscan.

***Примечание:*** *Чтобы выключить все функции, использующие hyperscan, используйте настройку SET allow\_hyperscan = 0.*

###### extract(haystack, pattern)

Извлечение фрагмента строки по регулярному выражению. Если haystack не соответствует регулярному выражению pattern, то возвращается пустая строка. Если регулярное выражение не содержит subpattern-ов, то вынимается фрагмент, который подпадает под всё регулярное выражение. Иначе вынимается фрагмент, который подпадает под первый subpattern.

###### extractAll(haystack, pattern)

Извлечение всех фрагментов строки по регулярному выражению. Если haystack не соответствует регулярному выражению pattern, то возвращается пустая строка. Возвращается массив строк, состоящий из всех соответствий регулярному выражению. В остальном, поведение аналогично функции extract (по-прежнему, вынимается первый subpattern, или всё выражение, если subpattern-а нет).

###### extractAllGroupsHorizontal

Разбирает строку haystack на фрагменты, соответствующие группам регулярного выражения pattern. Возвращает массив массивов, где первый массив содержит все фрагменты, соответствующие первой группе регулярного выражения, второй массив - соответствующие второй группе, и т.д.

***Замечание:*** *Функция extractAllGroupsHorizontal работает медленнее, чем функция* [*extractAllGroupsVertical*](https://clickhouse.tech/docs/ru/sql-reference/functions/string-search-functions/#extractallgroups-vertical)*.*

Синтаксис:

extractAllGroupsHorizontal(haystack, pattern)

Параметры:

* haystack – строка для разбора. Тип: [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).
* pattern – регулярное выражение, построенное по синтаксическим правилам [re2](https://github.com/google/re2/wiki/Syntax). Выражение должно содержать группы, заключенные в круглые скобки. Если выражение не содержит групп, генерируется исключение. Тип: [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).

Возвращаемое значение:

* Тип: [Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/).

Если в строке haystack нет групп, соответствующих регулярному выражению pattern, возвращается массив пустых массивов.

Пример:

Запрос:

**SELECT** extractAllGroupsHorizontal('abc=111, def=222, ghi=333', '("[^"]+"|\\w+)=("[^"]+"|\\w+)')

Результат:

┌─extractAllGroupsHorizontal('abc=111, def=222, ghi=333', '("[^"]+"|\\w+)=("[^"]+"|\\w+)')─┐

│ [['abc','def','ghi'],['111','222','333']] │

└──────────────────────────────────────────────────────────────────────────────────────────┘

См. также:

* функция [extractAllGroupsVertical](https://clickhouse.tech/docs/ru/sql-reference/functions/string-search-functions/#extractallgroups-vertical).

###### extractAllGroupsVertical

Разбирает строку haystack на фрагменты, соответствующие группам регулярного выражения pattern. Возвращает массив массивов, где каждый массив содержит по одному фрагменту, соответствующему каждой группе регулярного выражения. Фрагменты группируются в массивы в соответствии с порядком появления в исходной строке.

Синтаксис:

extractAllGroupsVertical(haystack, pattern)

Параметры:

* haystack – строка для разбора. Тип: [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).
* pattern – регулярное выражение, построенное по синтаксическим правилам [re2](https://github.com/google/re2/wiki/Syntax). Выражение должно содержать группы, заключенные в круглые скобки. Если выражение не содержит групп, генерируется исключение. Тип: [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).

Возвращаемое значение:

* Тип: [Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/).

Если в строке haystack нет групп, соответствующих регулярному выражению pattern, возвращается пустой массив.

Пример:

Запрос:

**SELECT** extractAllGroupsVertical('abc=111, def=222, ghi=333', '("[^"]+"|\\w+)=("[^"]+"|\\w+)')

Результат:

┌─extractAllGroupsVertical('abc=111, def=222, ghi=333', '("[^"]+"|\\w+)=("[^"]+"|\\w+)')─┐

│ [['abc','111'],['def','222'],['ghi','333']] │

└────────────────────────────────────────────────────────────────────────────────────────┘

См. также:

* функция [extractAllGroupsHorizontal](https://clickhouse.tech/docs/ru/sql-reference/functions/string-search-functions/#extractallgroups-horizontal).

###### like(haystack, pattern), оператор haystack LIKE pattern

Проверка строки на соответствие простому регулярному выражению.

Регулярное выражение может содержать метасимволы % и \_.

% обозначает любое количество любых байт (в том числе, нулевое количество символов).

\_ обозначает один любой байт.

Для экранирования метасимволов, используется символ \ (обратный слеш). Смотрите замечание об экранировании в описании функции match.

Для регулярных выражений вида %needle% действует более оптимальный код, который работает также быстро, как функция position.

Для остальных регулярных выражений, код аналогичен функции match.

###### notLike(haystack, pattern), оператор haystack NOT LIKE pattern

То же, что like, но с отрицанием.

###### ilike

Нечувствительный к регистру вариант функции [like](https://clickhouse.tech/docs/ru/sql-reference/functions/string-search-functions/#function-like). Вы можете использовать оператор ILIKE вместо функции ilike.

Синтаксис:

**ilike**(haystack, pattern)

Параметры:

* haystack – входная строка. [String](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-string-literal).
* pattern – если pattern не содержит процента или нижнего подчеркивания, тогда pattern представляет саму строку. Нижнее подчеркивание (\_) в pattern обозначает любой отдельный символ. Знак процента (%) соответствует последовательности из любого количества символов: от нуля и более.

Некоторые примеры pattern:

'abc' ILIKE 'abc' true

'abc' ILIKE 'a%' true

'abc' ILIKE '\_b\_' true

'abc' ILIKE 'c' false

Возвращаемые значения:

* Правда, если строка соответствует pattern.
* Ложь, если строка не соответствует pattern.

Пример:

Входная таблица:

┌─id─┬─name─────┬─days─┐

│ 1 │ January │ 31 │

│ 2 │ February │ 29 │

│ 3 │ March │ 31 │

│ 4 │ April │ 30 │

└────┴──────────┴──────┘

Запрос:

**SELECT** \* **FROM** Months **WHERE** **ilike**(name, '%j%')

Результат:

┌─id─┬─name────┬─days─┐

│ 1 │ January │ 31 │

└────┴─────────┴──────┘

Смотрите также:

* [like](https://clickhouse.tech/docs/ru/sql-reference/functions/string-search-functions/#function-like).

###### ngramDistance(haystack, needle)

Вычисление 4-граммного расстояния между haystack и needle: считается симметрическая разность между двумя мультимножествами 4-грамм и нормализуется на сумму их мощностей. Возвращает число float от 0 до 1 – чем ближе к нулю, тем больше строки похожи друг на друга. Если константный needle или haystack больше чем 32КБ, кидается исключение. Если некоторые строки из неконстантного haystack или needle больше 32КБ, расстояние всегда равно единице.

Для поиска без учета регистра и/или в формате UTF-8 используйте функции ngramDistanceCaseInsensitive, ngramDistanceUTF8, ngramDistanceCaseInsensitiveUTF8.

###### ngramSearch(haystack, needle)

То же, что и ngramDistance, но вычисляет несимметричную разность между needle и haystack – количество n-грамм из needle минус количество общих n-грамм, нормированное на количество n-грамм из needle. Чем ближе результат к единице, тем вероятнее, что needle внутри haystack. Может быть использовано для приближенного поиска.

Для поиска без учета регистра и/или в формате UTF-8 используйте функции ngramSearchCaseInsensitive, ngramSearchUTF8, ngramSearchCaseInsensitiveUTF8.

***Примечание:*** *Для случая UTF-8 мы используем триграммное расстояние. Вычисление n-граммного расстояния не совсем честное. Мы используем 2-х байтные хэши для хэширования n-грамм, а затем вычисляем (не)симметрическую разность между хэш таблицами – могут возникнуть коллизии. В формате UTF-8 без учета регистра мы не используем честную функцию tolower – мы обнуляем 5-й бит (нумерация с нуля) каждого байта кодовой точки, а также первый бит нулевого байта, если байтов больше 1 – это работает для латиницы и почти для всех кириллических букв.*

###### countMatches(haystack, pattern)

Возвращает количество совпадений, найденных в строке haystack, для регулярного выражения pattern.

Синтаксис:

countMatches(haystack, pattern)

Параметры:

* haystack – строка, по которой выполняется поиск. [String](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-string-literal).
* pattern – регулярное выражение, построенное по синтаксическим правилам [re2](https://github.com/google/re2/wiki/Syntax). [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).

Возвращаемое значение:

* Количество совпадений.

Тип: [UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Примеры:

Запрос:

**SELECT** countMatches('foobar.com', 'o+');

Результат:

┌─countMatches('foobar.com', 'o+')─┐

│ 2 │

└──────────────────────────────────┘

Запрос:

**SELECT** countMatches('aaaa', 'aa');

Результат:

┌─countMatches('aaaa', 'aa')────┐

│ 2 │

└───────────────────────────────┘

###### countSubstrings

Возвращает количество вхождений подстроки.

Для поиска без учета регистра, используйте функции [countSubstringsCaseInsensitive](https://clickhouse.tech/docs/ru/sql-reference/functions/string-search-functions/#countSubstringsCaseInsensitive) или [countSubstringsCaseInsensitiveUTF8](https://clickhouse.tech/docs/ru/sql-reference/functions/string-search-functions/#countSubstringsCaseInsensitiveUTF8)

Синтаксис:

countSubstrings(haystack, needle[, start\_pos])

Параметры:

* haystack – строка, в которой ведется поиск. [String](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-string-literal).
* needle – искомая подстрока. [String](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-string-literal).
* start\_pos – позиция первого символа в строке, с которого начнется поиск. Необязательный параметр. [UInt](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Возвращаемые значения:

* Число вхождений.

Тип: [UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Примеры:

Запрос:

**SELECT** countSubstrings('foobar.com', '.');

Результат:

┌─countSubstrings('foobar.com', '.')─┐

│ 1 │

└────────────────────────────────────┘

Запрос:

**SELECT** countSubstrings('aaaa', 'aa');

Результат:

┌─countSubstrings('aaaa', 'aa')─┐

│ 2 │

└───────────────────────────────┘

Запрос:

**SELECT** countSubstrings('abc\_\_\_abc', 'abc', 4);

Результат:

┌─countSubstrings('abc\_\_\_abc', 'abc', 4)─┐

│ 1 │

└────────────────────────────────────────┘

###### countSubstringsCaseInsensitive

Возвращает количество вхождений подстроки без учета регистра.

Синтаксис:

countSubstringsCaseInsensitive(haystack, needle[, start\_pos])

Параметры:

* haystack – строка, в которой ведется поиск. [String](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-string-literal).
* needle – искомая подстрока. [String](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-string-literal).
* start\_pos – позиция первого символа в строке, с которого начнется поиск. Необязательный параметр. [UInt](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Возвращаемые значения:

* Число вхождений.

Тип: [UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Примеры:

Запрос:

**select** countSubstringsCaseInsensitive('aba', 'B');

Результат:

┌─countSubstringsCaseInsensitive('aba', 'B')─┐

│ 1 │

└────────────────────────────────────────────┘

Запрос:

**SELECT** countSubstringsCaseInsensitive('foobar.com', 'CoM');

Результат:

┌─countSubstringsCaseInsensitive('foobar.com', 'CoM')─┐

│ 1 │

└─────────────────────────────────────────────────────┘

Запрос:

**SELECT** countSubstringsCaseInsensitive('abC\_\_\_abC', 'aBc', 2);

Результат:

┌─countSubstringsCaseInsensitive('abC\_\_\_abC', 'aBc', 2)─┐

│ 1 │

└───────────────────────────────────────────────────────┘

###### countSubstringsCaseInsensitiveUTF8

Возвращает количество вхождений подстроки в UTF-8 без учета регистра.

Синтаксис:

**SELECT** countSubstringsCaseInsensitiveUTF8(haystack, needle[, start\_pos])

Параметры:

* haystack – строка, в которой ведется поиск. [String](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-string-literal).
* needle – искомая подстрока. [String](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-string-literal).
* start\_pos – позиция первого символа в строке, с которого начнется поиск. Необязательный параметр. [UInt](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Возвращаемые значения:

* Число вхождений.

Тип: [UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Примеры:

Запрос:

**SELECT** countSubstringsCaseInsensitiveUTF8('абв', 'A');

Результат:

┌─countSubstringsCaseInsensitiveUTF8('абв', 'A')─┐

│ 1 │

└────────────────────────────────────────────────┘

Запрос:

**SELECT** countSubstringsCaseInsensitiveUTF8('аБв\_\_АбВ\_\_абв', 'Абв');

Результат:

┌─countSubstringsCaseInsensitiveUTF8('аБв\_\_АбВ\_\_абв', 'Абв')─┐

│ 3 │

└────────────────────────────────────────────────────────────┘

##### Функции поиска и замены в строках

###### replaceOne(haystack, pattern, replacement)

Замена первого вхождения, если такое есть, подстроки pattern в haystack на подстроку replacement.

Здесь и далее, pattern и replacement должны быть константами.

###### replaceAll(haystack, pattern, replacement)

Замена всех вхождений подстроки pattern в haystack на подстроку replacement.

###### replaceRegexpOne(haystack, pattern, replacement)

Замена по регулярному выражению pattern. Регулярное выражение re2.

Заменяется только первое вхождение, если есть.

В качестве replacement может быть указан шаблон для замен. Этот шаблон может включать в себя подстановки \0-\9.

Подстановка \0 - вхождение регулярного выражения целиком. Подстановки \1-\9 - соответствующие по номеру subpattern-ы.

Для указания символа \ в шаблоне, он должен быть экранирован с помощью символа \.

Также помните о том, что строковый литерал требует ещё одно экранирование.

Пример 1. Переведём дату в американский формат:

**SELECT** **DISTINCT**

EventDate,

replaceRegexpOne(toString(EventDate), '(\\d{4})-(\\d{2})-(\\d{2})', '\\2/\\3/\\1') **AS** res

**FROM** test.hits

**LIMIT** 7

FORMAT TabSeparated

2014-03-17 03/17/2014

2014-03-18 03/18/2014

2014-03-19 03/19/2014

2014-03-20 03/20/2014

2014-03-21 03/21/2014

2014-03-22 03/22/2014

2014-03-23 03/23/2014

Пример 2. Размножить строку десять раз:

**SELECT** replaceRegexpOne('Hello, World!', '.\*', '\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0') **AS** res

┌─res────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────┐

│ Hello, World!Hello, World!Hello, World!Hello, World!Hello, World!Hello, World!Hello, World!Hello, World!Hello, World!Hello, World! │

└────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────┘

###### replaceRegexpAll(haystack, pattern, replacement)

То же самое, но делается замена всех вхождений. Пример:

**SELECT** replaceRegexpAll('Hello, World!', '.', '\\0\\0') **AS** res

┌─res────────────────────────┐

│ HHeelllloo,, WWoorrlldd!! │

└────────────────────────────┘

В качестве исключения, если регулярное выражение сработало на пустой подстроке, то замена делается не более одного раза.

Пример:

**SELECT** replaceRegexpAll('Hello, World!', '^', 'here: ') **AS** res

┌─res─────────────────┐

│ here: Hello, World! │

└─────────────────────┘

##### Условные функции

###### if

Условное выражение. В отличии от большинства систем, RT.WideStore всегда считает оба выражения then и else.

Синтаксис:

**SELECT** **if**(cond, **then**, **else**)

Если условие cond не равно нулю, то возвращается результат выражения then. Если условие cond равно нулю или является NULL, то результат выражения then пропускается и возвращается результат выражения else.

Параметры:

* cond – условие, которое может быть равно 0 или нет. Может быть [UInt8](https://clickhouse.tech/docs/ru/sql-reference/functions/conditional-functions/) или NULL.
* then – возвращается результат выражения, если условие cond истинно.
* else – возвращается результат выражения, если условие cond ложно.

Возвращаемые значения:

Функция выполняет выражения then или else и возвращает его результат, в зависимости от того, было ли условие cond равно нулю или нет.

Пример:

Запрос:

**SELECT** **if**(1, plus(2, 2), plus(2, 6))

Ответ:

┌─plus(2, 2)─┐

│ 4 │

└────────────┘

Запрос:

**SELECT** **if**(0, plus(2, 2), plus(2, 6))

Ответ:

┌─plus(2, 6)─┐

│ 8 │

└────────────┘

###### Тернарный оператор

Работает так же, как функция if.

Синтаксис: cond ? then : else.

Возвращает then, если cond верно (больше нуля), в остальных случаях возвращает else.

* cond должно быть типа UInt8, then и else должны относиться к наименьшему общему типу.
* then и else могут быть NULL.

Смотрите также

* [ifNotFinite](https://clickhouse.tech/docs/ru/sql-reference/functions/other-functions/#ifnotfinite).

###### multiIf

Позволяет более компактно записать оператор [CASE](https://clickhouse.tech/docs/ru/sql-reference/operators/#operator_case) в запросе.

Синтаксис:

multiIf(cond\_1, then\_1, cond\_2, then\_2...else)

Параметры:

* cond\_N – условие, при выполнении которого функция вернёт then\_N.
* then\_N – результат функции при выполнении.
* else – результат функции, если ни одно из условий не выполнено.

Функция принимает 2N+1 параметров.

Возвращаемые значения:

* Функция возвращает одно из значений then\_N или else, в зависимости от условий cond\_N.

Пример:

Рассмотрим таблицу:

┌─x─┬────y─┐

│ 1 │ ᴺᵁᴸᴸ │

│ 2 │ 3 │

└───┴──────┘

Выполним запрос SELECT multiIf(isNull(y), x, y < 3, y, NULL) FROM t\_null. Результат:

┌─multiIf(isNull(y), x, less(y, 3), y, NULL)─┐

│ 1 │

│ ᴺᵁᴸᴸ │

└────────────────────────────────────────────┘

##### Математические функции

Все функции возвращают число типа Float64. Точность результата близка к максимально возможной, но результат может не совпадать с наиболее близким к соответствующему вещественному числу машинно представимым числом.

###### e()

Возвращает число типа Float64, близкое к числу e.

###### pi()

Возвращает число типа Float64, близкое к числу π.

###### exp(x)

Принимает числовой аргумент, возвращает число типа Float64, близкое к экспоненте от аргумента.

###### log(x)

Принимает числовой аргумент, возвращает число типа Float64, близкое к натуральному логарифму от аргумента.

###### exp2(x)

Принимает числовой аргумент, возвращает число типа Float64, близкое к 2 в степени x.

###### log2(x)

Принимает числовой аргумент, возвращает число типа Float64, близкое к двоичному логарифму от аргумента.

###### exp10(x)

Принимает числовой аргумент, возвращает число типа Float64, близкое к 10 в степени x.

###### log10(x)

Принимает числовой аргумент, возвращает число типа Float64, близкое к десятичному логарифму от аргумента.

###### sqrt(x)

Принимает числовой аргумент, возвращает число типа Float64, близкое к квадратному корню от аргумента.

###### cbrt(x)

Принимает числовой аргумент, возвращает число типа Float64, близкое к кубическому корню от аргумента.

###### erf(x)

Если x неотрицательно, то erf(x / σ√2) - вероятность того, что случайная величина, имеющая нормальное распределение со среднеквадратичным отклонением σ, принимает значение, отстоящее от мат. ожидания больше чем на x.

Пример (правило трёх сигм):

**SELECT** erf(3 / sqrt(2))

┌─erf(divide(3, sqrt(2)))─┐

│ 0.9973002039367398 │

└─────────────────────────┘

###### erfc(x)

Принимает числовой аргумент, возвращает число типа Float64, близкое к 1 - erf(x), но без потери точности для больших x.

###### lgamma(x)

Логарифм от гамма функции.

###### tgamma(x)

Гамма функция.

###### sin(x)

Синус.

###### cos(x)

Косинус.

###### tan(x)

Тангенс.

###### asin(x)

Арксинус.

###### acos(x)

Арккосинус.

###### atan(x)

Арктангенс.

###### pow(x, y)

Принимает два числовых аргумента x и y. Возвращает число типа Float64, близкое к x в степени y.

###### cosh(x)

[Гиперболический косинус](https://help.scilab.org/docs/5.4.0/ru_RU/cosh.html).

Синтаксис:

cosh(x)

Параметры:

* x – угол в радианах. Значения из интервала: -∞ < x < +∞. [Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/#float32-float64).

Возвращаемое значение:

* Значения из интервала: 1 <= cosh(x) < +∞.

Тип: [Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/#float32-float64).

Пример:

Запрос:

**SELECT** cosh(0);

Результат:

┌─cosh(0)──┐

│ 1 │

└──────────┘

###### acosh(x)

[Обратный гиперболический косинус](https://help.scilab.org/docs/5.4.0/ru_RU/acosh.html).

Синтаксис:

acosh(x)

Параметры:

* x – гиперболический косинус угла. Значения из интервала: 1 <= x < +∞. [Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/#float32-float64).

Возвращаемое значение:

* Угол в радианах. Значения из интервала: 0 <= acosh(x) < +∞.

Тип: [Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/#float32-float64).

Пример:

Запрос:

**SELECT** acosh(1);

Результат:

┌─acosh(1)─┐

│ 0 │

└──────────┘

Смотрите также:

* [cosh(x)](https://clickhouse.tech/docs/ru/sql-reference/functions/math-functions/#coshx).

###### sinh(x)

[Гиперболический синус](https://help.scilab.org/docs/5.4.0/ru_RU/sinh.html).

Синтаксис:

sinh(x)

Параметры:

* x – угол в радианах. Значения из интервала: -∞ < x < +∞. [Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/#float32-float64).

Возвращаемое значение:

* Значения из интервала: -∞ < sinh(x) < +∞.

Тип: [Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/#float32-float64).

Пример:

Запрос:

**SELECT** sinh(0);

Результат:

┌─sinh(0)──┐

│ 0 │

└──────────┘

###### asinh(x)

[Обратный гиперболический синус](https://help.scilab.org/docs/5.4.0/ru_RU/asinh.html).

Синтаксис:

asinh(x)

Параметры:

* x – гиперболический синус угла. Значения из интервала: -∞ < x < +∞. [Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/#float32-float64).

Возвращаемое значение:

* Угол в радианах. Значения из интервала: -∞ < asinh(x) < +∞.

Тип: [Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/#float32-float64).

Пример:

Запрос:

**SELECT** asinh(0);

Результат:

┌─asinh(0)─┐

│ 0 │

└──────────┘

Смотрите также:

* [sinh(x)](https://clickhouse.tech/docs/ru/sql-reference/functions/math-functions/#sinhx).

###### atanh(x)

[Обратный гиперболический тангенс](https://help.scilab.org/docs/5.4.0/ru_RU/atanh.html).

Синтаксис:

atanh(x)

Параметры:

* x – гиперболический тангенс угла. Значения из интервала: –1 < x < 1. [Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/#float32-float64).

Возвращаемое значение:

* Угол в радианах. Значения из интервала: -∞ < atanh(x) < +∞.

Тип: [Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/#float32-float64).

Пример:

Запрос:

**SELECT** atanh(0);

Результат:

┌─atanh(0)─┐

│ 0 │

└──────────┘

###### atan2(y, x)

[Функция](https://msoffice-prowork.com/ref/excel/excelfunc/math/atan2/) вычисляет угол в радианах между положительной осью x и линией, проведенной из начала координат в точку (x, y) ≠ (0, 0).

Синтаксис:

atan2(y, x)

Параметры:

* y – координата y точки, в которую проведена линия. [Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/#float32-float64).
* x – координата х точки, в которую проведена линия. [Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/#float32-float64).

Возвращаемое значение:

* Угол θ в радианах из интервала: −π < θ ≤ π.

Тип: [Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/#float32-float64).

Пример:

Запрос:

**SELECT** atan2(1, 1);

Результат:

┌────────atan2(1, 1)─┐

│ 0.7853981633974483 │

└────────────────────┘

###### hypot(x, y)

Вычисляет длину гипотенузы прямоугольного треугольника. При использовании этой [функции](https://php.ru/manual/function.hypot.html) не возникает проблем при возведении в квадрат очень больших или очень малых чисел.

Синтаксис:

hypot(x, y)

Параметры:

* x – первый катет прямоугольного треугольника. [Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/#float32-float64).
* y – второй катет прямоугольного треугольника. [Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/#float32-float64).

Возвращаемое значение:

* Длина гипотенузы прямоугольного треугольника.

Тип: [Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/#float32-float64).

Пример:

Запрос:

**SELECT** hypot(1, 1);

Результат:

┌────────hypot(1, 1)─┐

│ 1.4142135623730951 │

└────────────────────┘

###### log1p(x)

Вычисляет log(1+x). [Функция](https://help.scilab.org/docs/6.0.1/ru_RU/log1p.html) log1p(x) является более точной, чем функция log(1+x) для малых значений x.

Синтаксис:

log1p(x)

Параметры:

* x – значения из интервала: -1 < x < +∞. [Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/#float32-float64).

Возвращаемое значение:

* Значения из интервала: -∞ < log1p(x) < +∞.

Тип: [Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/#float32-float64).

Пример:

Запрос:

**SELECT** log1p(0);

Результат:

┌─log1p(0)─┐

│ 0 │

└──────────┘

Смотрите также:

* [log(x)](https://clickhouse.tech/docs/ru/sql-reference/functions/math-functions/#logx).

##### Функции округления

###### floor(x[, N])

Возвращает наибольшее круглое число, которое меньше или равно, чем x.

Круглым называется число, кратное 1 / 10N или ближайшее к нему число соответствующего типа данных, если 1 / 10N не представимо точно.

N – целочисленная константа, не обязательный параметр. По умолчанию – ноль, что означает – округлять до целого числа.

N может быть отрицательным.

Примеры: floor(123.45, 1) = 123.4, floor(123.45, -1) = 120.

x - любой числовой тип. Результат - число того же типа.

Для целочисленных аргументов имеет смысл округление с отрицательным значением N (для неотрицательных N, функция ничего не делает).

В случае переполнения при округлении (например, floor(-128, -1)), возвращается implementation specific результат.

###### ceil(x[, N])

Возвращает наименьшее круглое число, которое больше или равно, чем x.

В остальном, аналогично функции floor, см. выше.

###### round(x[, N])

Округляет значение до указанного десятичного разряда.

Функция возвращает ближайшее значение указанного порядка. В случае, когда заданное число равноудалено от чисел необходимого порядка, функция возвращает то из них, которое имеет ближайшую чётную цифру (банковское округление).

Синтаксис:

round(expression [, decimal\_places])

Параметры:

* expression – Число для округления. Может быть любым [выражением](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-expressions), возвращающим числовой [тип данных](https://clickhouse.tech/docs/ru/sql-reference/functions/rounding-functions/#data_types).
* decimal-places – Целое значение:
  + Если decimal-places > 0, то функция округляет значение справа от запятой.
  + Если decimal-places < 0 то функция округляет значение слева от запятой.
  + Если decimal-places = 0, то функция округляет значение до целого. В этом случае аргумент можно опустить.

Возвращаемое значение:

Округлённое значение того же типа, что и входящее.

Примеры:

Пример использования:

**SELECT** number / 2 **AS** x, round(x) **FROM** **system**.numbers **LIMIT** 3

┌───x─┬─round(divide(number, 2))─┐

│ 0 │ 0 │

│ 0.5 │ 0 │

│ 1 │ 1 │

└─────┴──────────────────────────┘

Примеры округления:

Округление до ближайшего числа.

round(3.2, 0) = 3

round(4.1267, 2) = 4.13

round(22,-1) = 20

round(467,-2) = 500

round(-467,-2) = -500

Банковское округление.

round(3.5) = 4

round(4.5) = 4

round(3.55, 1) = 3.6

round(3.65, 1) = 3.6

Смотрите также:

* [roundBankers](https://clickhouse.tech/docs/ru/sql-reference/functions/rounding-functions/#roundbankers).

###### roundBankers

Округляет число до указанного десятичного разряда.

* Если округляемое число равноудалено от соседних чисел, то используется банковское округление.

Банковское округление (англ. banker's rounding) – метод округления дробных чисел. Если округляемое число равноудалено от соседних чисел, то оно округляется до ближайшей чётной цифры заданного десятичного разряда. К примеру, 3,5 округляется до 4, а 2,5 до 2.

Этот метод округления, используемый по умолчанию для чисел с плавающей запятой, определён в стандарте [IEEE 754](https://en.wikipedia.org/wiki/IEEE\_754#Roundings\_to\_nearest). Функция [round](#rounding\_functions-round) также округляет числа с плавающей запятой по этому методу. Функция `roundBankers` округляет не только числа с плавающей запятой, но и целые числа методом банковского округления, например, `roundBankers(45, -1) = 40`.

В других случаях функция округляет к ближайшему целому.

Банковское округление позволяет уменьшить влияние округления чисел на результат суммирования или вычитания этих чисел.

Пример суммирования чисел 1.5, 2.5, 3.5 и 4.5 с различным округлением:

* Без округления: 1.5 + 2.5 + 3.5 + 4.5 = 12.
* Банковское округление: 2 + 2 + 4 + 4 = 12.
* Округление до ближайшего целого: 2 + 3 + 4 + 5 = 14.

Синтаксис:

roundBankers(expression [, decimal\_places])

Параметры:

* expression – число для округления. Может быть любым [выражением](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-expressions), возвращающим числовой [тип данных](https://clickhouse.tech/docs/ru/sql-reference/functions/rounding-functions/#data_types).
* decimal-places – десятичный разряд. Целое число:
  + decimal-places > 0 – функция округляет значение выражения до ближайшего чётного числа на соответствующей позиции справа от запятой. Например, roundBankers(3.55, 1) = 3.6.
  + decimal-places < 0 – функция округляет значение выражения до ближайшего чётного числа на соответствующей позиции слева от запятой. Например, roundBankers(24.55, -1) = 20.
  + decimal-places = 0 – функция округляет значение до целого. В этом случае аргумент можно не передавать. Например, roundBankers(2.5) = 2.

Возвращаемое значение:

* Округлённое значение по методу банковского округления.

Пример использования:

Запрос:

**SELECT** number / 2 **AS** x, roundBankers(x, 0) **AS** b **fROM** **system**.numbers **limit** 10

Результат:

┌───x─┬─b─┐

│ 0 │ 0 │

│ 0.5 │ 0 │

│ 1 │ 1 │

│ 1.5 │ 2 │

│ 2 │ 2 │

│ 2.5 │ 2 │

│ 3 │ 3 │

│ 3.5 │ 4 │

│ 4 │ 4 │

│ 4.5 │ 4 │

└─────┴───┘

Примеры банковского округления:

roundBankers(0.4) = 0

roundBankers(-3.5) = -4

roundBankers(4.5) = 4

roundBankers(3.55, 1) = 3.6

roundBankers(3.65, 1) = 3.6

roundBankers(10.35, 1) = 10.4

roundBankers(10.755, 2) = 11,76

Смотрите также:

* [round](https://clickhouse.tech/docs/ru/sql-reference/functions/rounding-functions/#rounding_functions-round).

###### roundToExp2(num)

Принимает число. Если число меньше единицы - возвращает 0. Иначе округляет число вниз до ближайшей (целой неотрицательной) степени двух.

###### roundDuration(num)

Принимает число. Если число меньше единицы - возвращает 0. Иначе округляет число вниз до чисел из набора: 1, 10, 30, 60, 120, 180, 240, 300, 600, 1200, 1800, 3600, 7200, 18000, 36000. Эта функция специфична и предназначена для реализации отчёта по длительности визита.

###### roundAge(num)

Принимает число. Если число меньше 18 - возвращает 0. Иначе округляет число вниз до чисел из набора: 18, 25, 35, 45, 55. Эта функция специфична и предназначена для реализации отчёта по возрасту посетителей.

##### Функции для работы с контейнерами map

###### map

Преобразовывает пары ключ:значение в тип данных [Map(key, value)](https://clickhouse.tech/docs/ru/sql-reference/data-types/map/).

Синтаксис:

**map**(key1, value1[, key2, value2, ...])

Параметры:

* key – ключ. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/) или [Integer](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).
* value – значение. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/), [Integer](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/) или [Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/).

Возвращаемое значение:

* Структура данных в виде пар ключ: значение.

Тип: [Map(key, value)](https://clickhouse.tech/docs/ru/sql-reference/data-types/map/).

Примеры:

Запрос:

**SELECT** **map**('key1', number, 'key2', number \* 2) **FROM** numbers(3);

Результат:

┌─map('key1', number, 'key2', multiply(number, 2))─┐

│ {'key1':0,'key2':0} │

│ {'key1':1,'key2':2} │

│ {'key1':2,'key2':4} │

└──────────────────────────────────────────────────┘

Запрос:

**CREATE** **TABLE** table\_map (a **Map**(String, UInt64)) ENGINE = MergeTree() **ORDER** **BY** a;

**INSERT** **INTO** table\_map **SELECT** **map**('key1', number, 'key2', number \* 2) **FROM** numbers(3);

**SELECT** a['key2'] **FROM** table\_map;

Результат:

┌─arrayElement(a, 'key2')─┐

│ 0 │

│ 2 │

│ 4 │

└─────────────────────────┘

См. также

* тип данных [Map(key, value)](https://clickhouse.tech/docs/ru/sql-reference/data-types/map/).

###### mapAdd

Собирает все ключи и суммирует соответствующие значения.

Синтаксис:

mapAdd(Tuple(Array, Array), Tuple(Array, Array) [, ...])

Параметры

Аргументами являются [кортежи](https://clickhouse.tech/docs/ru/sql-reference/data-types/tuple/#tuplet1-t2) из двух [массивов](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/#data-type-array), где элементы в первом массиве представляют ключи, а второй массив содержит значения для каждого ключа.

Все массивы ключей должны иметь один и тот же тип, а все массивы значений должны содержать элементы, которые можно приводить к одному типу ([Int64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#int-ranges), [UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges) или [Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/#float32-float64)).

Общий приведенный тип используется в качестве типа для результирующего массива.

Возвращаемое значение:

* Возвращает один [кортеж](https://clickhouse.tech/docs/ru/sql-reference/data-types/tuple/#tuplet1-t2), в котором первый массив содержит отсортированные ключи, а второй – значения.

Пример:

Запрос:

**SELECT** mapAdd(([toUInt8(1), 2], [1, 1]), ([toUInt8(1), 2], [1, 1])) **as** res, toTypeName(res) **as** **type**;

Результат:

┌─res───────────┬─type───────────────────────────────┐

│ ([1,2],[2,2]) │ Tuple(Array(UInt8), Array(UInt64)) │

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

###### mapSubtract

Собирает все ключи и вычитает соответствующие значения.

Синтаксис:

mapSubtract(Tuple(Array, Array), Tuple(Array, Array) [, ...])

Параметры:

Аргументами являются [кортежи](https://clickhouse.tech/docs/ru/sql-reference/data-types/tuple/#tuplet1-t2) из двух [массивов](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/#data-type-array), где элементы в первом массиве представляют ключи, а второй массив содержит значения для каждого ключа.  
Все массивы ключей должны иметь один и тот же тип, а все массивы значений должны содержать элементы, которые можно приводить к одному типу ([Int64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#int-ranges), [UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges) или [Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/#float32-float64)).  
Общий приведенный тип используется в качестве типа для результирующего массива.

Возвращаемое значение:

* Возвращает один [tuple](https://clickhouse.tech/docs/ru/sql-reference/data-types/tuple/#tuplet1-t2), в котором первый массив содержит отсортированные ключи, а второй – значения.

Пример:

Запрос:

**SELECT** mapSubtract(([toUInt8(1), 2], [toInt32(1), 1]), ([toUInt8(1), 2], [toInt32(2), 1])) **as** res, toTypeName(res) **as** **type**;

Результат:

┌─res────────────┬─type──────────────────────────────┐

│ ([1,2],[-1,0]) │ Tuple(Array(UInt8), Array(Int64)) │

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

###### mapPopulateSeries

Заполняет недостающие ключи в контейнере map (пара массивов ключей и значений), где ключи являются целыми числами. Кроме того, он поддерживает указание максимального ключа, который используется для расширения массива ключей.

Синтаксис:

mapPopulateSeries(keys, **values**[, **max**])

Генерирует контейнер map, где ключи - это серия чисел, от минимального до максимального ключа (или аргумент max, если он указан), взятых из массива keys с размером шага один, и соответствующие значения, взятые из массива values. Если значение не указано для ключа, то в результирующем контейнере используется значение по умолчанию.

Количество элементов в keys и values должно быть одинаковым для каждой строки.

Параметры:

* keys – массив ключей [Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/#data-type-array)([Int](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#int-ranges)).
* values – массив значений. [Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/#data-type-array)([Int](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#int-ranges)).

Возвращаемое значение:

* Возвращает [кортеж](https://clickhouse.tech/docs/ru/sql-reference/data-types/tuple/#tuplet1-t2) из двух [массивов](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/#data-type-array): ключи отсортированные по порядку и значения соответствующих ключей.

Пример:

Запрос:

**select** mapPopulateSeries([1,2,4], [11,22,44], 5) **as** res, toTypeName(res) **as** **type**;

Результат:

┌─res──────────────────────────┬─type──────────────────────────────┐

│ ([1,2,3,4,5],[11,22,0,44,0]) │ Tuple(Array(UInt8), Array(UInt8)) │

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

###### mapContains

Определяет, содержит ли контейнер map ключ key.

Синтаксис:

mapContains(**map**, **key**)

Параметры:

* map – контейнер Map. [Map](https://clickhouse.tech/docs/ru/sql-reference/data-types/map/).
* key – ключ. Тип соответстует типу ключей параметра map.

Возвращаемое значение:

* 1 если map включает key, иначе 0.

Тип: [UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Пример:

Запрос:

**CREATE** **TABLE** test (a **Map**(String,String)) ENGINE = Memory;

**INSERT** **INTO** test **VALUES** ({'name':'eleven','age':'11'}), ({'number':'twelve','position':'6.0'});

**SELECT** mapContains(a, 'name') **FROM** test;

Результат:

┌─mapContains(a, 'name')─┐

│ 1 │

│ 0 │

└────────────────────────┘

###### mapKeys

Возвращает все ключи контейнера map.

Синтаксис:

mapKeys(**map**)

Параметры

* map – контейнер Map. [Map](https://clickhouse.tech/docs/ru/sql-reference/data-types/map/).

Возвращаемое значение

* Массив со всеми ключами контейнера map.

Тип: [Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/).

Пример:

Запрос:

**CREATE** **TABLE** test (a **Map**(String,String)) ENGINE = Memory;

**INSERT** **INTO** test **VALUES** ({'name':'eleven','age':'11'}), ({'number':'twelve','position':'6.0'});

**SELECT** mapKeys(a) **FROM** test;

Результат:

┌─mapKeys(a)────────────┐

│ ['name','age'] │

│ ['number','position'] │

└───────────────────────┘

###### mapValues

Возвращает все значения контейнера map.

Синтаксис:

mapKeys(**map**)

Параметры:

* map – контейнер Map. [Map](https://clickhouse.tech/docs/ru/sql-reference/data-types/map/).

Возвращаемое значение:

* Массив со всеми значениями контейнера map.

Тип: [Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/).

Примеры:

Запрос:

**CREATE** **TABLE** test (a **Map**(String,String)) ENGINE = Memory;

**INSERT** **INTO** test **VALUES** ({'name':'eleven','age':'11'}), ({'number':'twelve','position':'6.0'});

**SELECT** mapValues(a) **FROM** test;

Результат:

┌─mapValues(a)─────┐

│ ['eleven','11'] │

│ ['twelve','6.0'] │

└──────────────────┘

##### Функции разбиения и слияния строк и массивов

###### splitByChar(separator, s)

Разбивает строку на подстроки, используя в качестве разделителя separator.  
separator должен быть константной строкой из ровно одного символа.  
Возвращается массив выделенных подстрок. Могут выделяться пустые подстроки, если разделитель идёт в начале или в конце строки, или если идёт более одного разделителя подряд.

Синтаксис:

splitByChar(<separator>, <s>)

Параметры:

* separator – разделитель, состоящий из одного символа. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).
* s – разбиваемая строка. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).

Возвращаемые значения:

Возвращает массив подстрок. Пустая подстрока, может быть возвращена, когда:

* Разделитель находится в начале или конце строки;
* Задано несколько последовательных разделителей;
* Исходная строка s пуста.

Type: [Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/) of [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).

Пример:

**SELECT** splitByChar(',', '1,2,3,abcde')

┌─splitByChar(',', '1,2,3,abcde')─┐

│ ['1','2','3','abcde'] │

└─────────────────────────────────┘

###### splitByString(separator, s)

Разбивает строку на подстроки, разделенные строкой. В качестве разделителя использует константную строку separator, которая может состоять из нескольких символов. Если строка separator пуста, то функция разделит строку s на массив из символов.

Синтаксис:

splitByString(separator, s)

Параметры:

* separator – разделитель. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).
* s – разбиваемая строка. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).

Возвращаемые значения:

Возвращает массив подстрок. Пустая подстрока, может быть возвращена, когда:

* Разделитель находится в начале или конце строки;
* Задано несколько последовательных разделителей;
* Исходная строка s пуста.

Type: [Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/) of [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).

Примеры:

**SELECT** splitByString(', ', '1, 2 3, 4,5, abcde')

┌─splitByString(', ', '1, 2 3, 4,5, abcde')─┐

│ ['1','2 3','4,5','abcde'] │

└───────────────────────────────────────────┘

**SELECT** splitByString('', 'abcde')

┌─splitByString('', 'abcde')─┐

│ ['a','b','c','d','e'] │

└────────────────────────────┘

###### arrayStringConcat(arr[, separator])

Склеивает строки, перечисленные в массиве, с разделителем separator.

separator – необязательный параметр, константная строка, по умолчанию равен пустой строке.

Возвращается строка.

###### alphaTokens(s)

Выделяет подстроки из подряд идущих байт из диапазонов a-z и A-Z.  
Возвращается массив выделенных подстрок.

Пример:

**SELECT** alphaTokens('abca1abc')

┌─alphaTokens('abca1abc')─┐

│ ['abca','abc'] │

└─────────────────────────┘

##### Битовые функции

Битовые функции работают для любой пары типов из UInt8, UInt16, UInt32, UInt64, Int8, Int16, Int32, Int64, Float32, Float64.

Тип результата – целое число, битность которого равна максимальной битности аргументов. Если хотя бы один аргумент знаковый, то результат – знаковое число. Если аргумент – число с плавающей запятой - оно приводится к Int64.

###### bitAnd(a, b)

###### bitOr(a, b)

###### bitXor(a, b)

###### bitNot(a)

###### bitShiftLeft(a, b)

###### bitShiftRight(a, b)

###### bitTest

Принимает любое целое число и конвертирует его в [двоичное число](https://en.wikipedia.org/wiki/Binary_number), возвращает значение бита в указанной позиции. Отсчет начинается с 0 справа налево.

Синтаксис:

**SELECT** bitTest(number, **index**)

Параметры:

* number – целое число.
* index – position of bit.

Возвращаемое значение:

* Возвращает значение бита в указанной позиции.

Тип: UInt8.

Пример:

Например, число 43 в двоичной системе счисления равно: 101011.

Запрос:

**SELECT** bitTest(43, 1)

Ответ:

┌─bitTest(43, 1)─┐

│ 1 │

└────────────────┘

Другой пример:

Запрос:

**SELECT** bitTest(43, 2)

Ответ:

┌─bitTest(43, 2)─┐

│ 0 │

└────────────────┘

###### bitTestAll

Возвращает результат [логической конъюнкции](https://en.wikipedia.org/wiki/Logical_conjunction) (оператор AND) всех битов в указанных позициях. Отсчет начинается с 0 справа налево.

Бинарная конъюнкция:

0 AND 0 = 0

0 AND 1 = 0

1 AND 0 = 0

1 AND 1 = 1

Синтаксис:

**SELECT** bitTestAll(number, index1, index2, index3, index4, ...)

Параметры:

* number – целое число.
* index1, index2, index3, index4 – позиция бита. Например, конъюнкция для набора позиций index1, index2, index3, index4 является истинной, если все его позиции истинны index1 ⋀ index2 ⋀ index3 ⋀ index4.

Возвращаемое значение:

* Возвращает результат логической конъюнкции.

Тип: UInt8.

Пример:

Например, число 43 в двоичной системе счисления равно: 101011.

Запрос:

**SELECT** bitTestAll(43, 0, 1, 3, 5)

Ответ:

┌─bitTestAll(43, 0, 1, 3, 5)─┐

│ 1 │

└────────────────────────────┘

Другой пример:

Запрос:

**SELECT** bitTestAll(43, 0, 1, 3, 5, 2)

Ответ:

┌─bitTestAll(43, 0, 1, 3, 5, 2)─┐

│ 0 │

└───────────────────────────────┘

###### bitTestAny

Возвращает результат [логической дизъюнкции](https://en.wikipedia.org/wiki/Logical_disjunction) (оператор OR) всех битов в указанных позициях. Отсчет начинается с 0 справа налево.

Бинарная дизъюнкция:

* 0 OR 0 = 0
* 0 OR 1 = 1
* 1 OR 0 = 1
* 1 OR 1 = 1

Синтаксис:

**SELECT** bitTestAny(number, index1, index2, index3, index4, ...)

Параметры:

* number – целое число.
* index1, index2, index3, index4 – позиции бита.

Возвращаемое значение:

* Возвращает результат логической дизъюнкции.

Тип: UInt8.

Пример:

Например, число 43 в двоичной системе счисления равно: 101011.

Запрос:

**SELECT** bitTestAny(43, 0, 2)

Ответ:

┌─bitTestAny(43, 0, 2)─┐

│ 1 │

└──────────────────────┘

Другой пример:

Запрос:

**SELECT** bitTestAny(43, 4, 2)

Ответ:

┌─bitTestAny(43, 4, 2)─┐

│ 0 │

└──────────────────────┘

###### bitCount

Подсчитывает количество равных единице бит в числе.

Синтаксис:

bitCount(x)

Параметры:

* x – [целое число](https://clickhouse.tech/docs/ru/sql-reference/functions/bit-functions/) или [число с плавающей запятой](https://clickhouse.tech/docs/ru/sql-reference/functions/bit-functions/). Функция использует представление числа в памяти, что позволяет поддержать числа с плавающей запятой.

Возвращаемое значение:

* Количество равных единице бит во входном числе.

Функция не преобразует входное значение в более крупный тип ([sign extension](https://en.wikipedia.org/wiki/Sign_extension)). Поэтому, например, bitCount(toUInt8(-1)) = 8.

Тип: UInt8.

Пример:

Возьмём к примеру число 333. Его бинарное представление – 0000000101001101.

Запрос:

**SELECT** bitCount(333)

Результат:

┌─bitCount(100)─┐

│ 5 │

└───────────────┘

##### Функции для битовых масок

###### bitmapBuild

Создаёт битовый массив из массива целочисленных значений.

Синтаксис:

bitmapBuild(array)

Параметры:

* array – массив типа UInt\*.

Пример:

**SELECT** bitmapBuild([1, 2, 3, 4, 5]) **AS** res, toTypeName(res)

┌─res─┬─toTypeName(bitmapBuild([1, 2, 3, 4, 5]))─────┐

│ │ AggregateFunction(groupBitmap, UInt8) │

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

###### bitmapToArray

Преобразует битовый массив в массив целочисленных значений.

Синтаксис:

bitmapToArray(bitmap)

Параметры:

* bitmap – битовый массив.

Пример:

**SELECT** bitmapToArray(bitmapBuild([1, 2, 3, 4, 5])) **AS** res

┌─res─────────┐

│ [1,2,3,4,5] │

└─────────────┘

###### bitmapSubsetLimit

Создает подмножество битмапа с n элементами, расположенными между range\_start и cardinality\_limit.

Синтаксис:

bitmapSubsetLimit(bitmap, range\_start, cardinality\_limit)

Параметры:

* bitmap – битмап. [Bitmap объект](https://clickhouse.tech/docs/ru/sql-reference/functions/bitmap-functions/#bitmap_functions-bitmapbuild).
* range\_start – начальная точка подмножества. [UInt32](https://clickhouse.tech/docs/ru/sql-reference/functions/bitmap-functions/#bitmap-functions).
* cardinality\_limit – Верхний предел подмножества. [UInt32](https://clickhouse.tech/docs/ru/sql-reference/functions/bitmap-functions/#bitmap-functions).

Возвращаемое значение:

* Подмножество битмапа.

Тип: Bitmap object.

Пример:

Запрос:

**SELECT** bitmapToArray(bitmapSubsetLimit(bitmapBuild([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,100,200,500]), toUInt32(30), toUInt32(200))) **AS** res

Ответ:

┌─res───────────────────────┐

│ [30,31,32,33,100,200,500] │

└───────────────────────────┘

###### bitmapContains

Проверяет вхождение элемента в битовый массив.

Синтаксис:

bitmapContains(haystack, needle)

Параметры:

* haystack – [объект Bitmap](https://clickhouse.tech/docs/ru/sql-reference/functions/bitmap-functions/#bitmap_functions-bitmapbuild), в котором функция ищет значение.
* needle – значение, которое функция ищет. Тип – [UInt32](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Возвращаемые значения:

* 0 – если в haystack нет needle.
* 1 – если в haystack есть needle.

Тип – UInt8.

Пример:

**SELECT** bitmapContains(bitmapBuild([1,5,7,9]), toUInt32(9)) **AS** res

┌─res─┐

│ 1 │

└─────┘

###### bitmapHasAny

Проверяет, имеют ли два битовых массива хотя бы один общий элемент.

Синтаксис:

bitmapHasAny(bitmap1, bitmap2)

Если вы уверены, что bitmap2 содержит строго один элемент, используйте функцию [bitmapContains](https://clickhouse.tech/docs/ru/sql-reference/functions/bitmap-functions/#bitmap_functions-bitmapcontains). Она работает эффективнее.

Параметры:

* bitmap\* – массив любого типа с набором элементов.

Возвращаемые значения:

* 1, если bitmap1 и bitmap2 имеют хотя бы один одинаковый элемент.
* 0, в противном случае.

Пример:

**SELECT** bitmapHasAny(bitmapBuild([1,2,3]),bitmapBuild([3,4,5])) **AS** res

┌─res─┐

│ 1 │

└─────┘

###### bitmapHasAll

Аналогично функции hasAll(array, array) возвращает 1 если первый битовый массив содержит все элементы второго, 0 в противном случае.

Если второй аргумент является пустым битовым массивом, то возвращает 1.

Синтаксис:

bitmapHasAll(bitmap,bitmap)

Параметры:

* bitmap – битовый массив.

Пример:

**SELECT** bitmapHasAll(bitmapBuild([1,2,3]),bitmapBuild([3,4,5])) **AS** res

┌─res─┐

│ 0 │

└─────┘

###### bitmapAnd

Логическое И для двух битовых массивов. Результат – новый битовый массив.

Синтаксис:

bitmapAnd(bitmap,bitmap)

Параметры:

* bitmap – битовый массив.

Пример:

**SELECT** bitmapToArray(bitmapAnd(bitmapBuild([1,2,3]),bitmapBuild([3,4,5]))) **AS** res

┌─res─┐

│ [3] │

└─────┘

###### bitmapOr

Логическое ИЛИ для двух битовых массивов. Результат – новый битовый массив.

Синтаксис:

bitmapOr(bitmap,bitmap)

Параметры:

* bitmap – битовый массив.

Пример:

**SELECT** bitmapToArray(bitmapOr(bitmapBuild([1,2,3]),bitmapBuild([3,4,5]))) **AS** res

┌─res─────────┐

│ [1,2,3,4,5] │

└─────────────┘

###### bitmapXor

Логическое исключающее ИЛИ для двух битовых массивов. Результат – новый битовый массив.

Синтаксис:

bitmapXor(bitmap,bitmap)

Параметры:

* bitmap – битовый массив.

Пример:

**SELECT** bitmapToArray(bitmapXor(bitmapBuild([1,2,3]),bitmapBuild([3,4,5]))) **AS** res

┌─res───────┐

│ [1,2,4,5] │

└───────────┘

###### bitmapAndnot

Логическое отрицание И для двух битовых массивов. Результат – новый битовый массив.

Синтаксис:

bitmapAndnot(bitmap,bitmap)

Параметры:

* bitmap – битовый массив.

Пример:

**SELECT** bitmapToArray(bitmapAndnot(bitmapBuild([1,2,3]),bitmapBuild([3,4,5]))) **AS** res

┌─res───┐

│ [1,2] │

└───────┘

###### bitmapCardinality

Возвращает кардинальность битового массива в виде значения типа UInt64.

Синтаксис:

bitmapCardinality(bitmap)

Параметры:

* bitmap – битовый массив.

Пример:

**SELECT** bitmapCardinality(bitmapBuild([1, 2, 3, 4, 5])) **AS** res

┌─res─┐

│ 5 │

└─────┘

###### bitmapAndCardinality

Выполняет логическое И и возвращает кардинальность (UInt64) результирующего битового массива.

Синтаксис:

bitmapAndCardinality(bitmap,bitmap)

Параметры:

* bitmap – битовый массив.

Пример:

**SELECT** bitmapAndCardinality(bitmapBuild([1,2,3]),bitmapBuild([3,4,5])) **AS** res;

┌─res─┐

│ 1 │

└─────┘

###### bitmapOrCardinality

Выполняет логическое ИЛИ и возвращает кардинальность (UInt64) результирующего битового массива.

Синтаксис:

bitmapOrCardinality(bitmap,bitmap)

Параметры:

* bitmap – битовый массив.

Пример:

**SELECT** bitmapOrCardinality(bitmapBuild([1,2,3]),bitmapBuild([3,4,5])) **AS** res;

┌─res─┐

│ 5 │

└─────┘

###### bitmapXorCardinality

Выполняет логическое исключающее ИЛИ и возвращает кардинальность (UInt64) результирующего битового массива.

Синтаксис:

bitmapXorCardinality(bitmap,bitmap)

Параметры:

* bitmap – битовый массив.

Пример:

**SELECT** bitmapXorCardinality(bitmapBuild([1,2,3]),bitmapBuild([3,4,5])) **AS** res;

┌─res─┐

│ 4 │

└─────┘

###### bitmapAndnotCardinality

Выполняет логическое отрицание И и возвращает кардинальность (UInt64) результирующего битового массива.

Синтаксис:

bitmapAndnotCardinality(bitmap,bitmap)

Параметры:

* bitmap – битовый массив.

Пример:

**SELECT** bitmapAndnotCardinality(bitmapBuild([1,2,3]),bitmapBuild([3,4,5])) **AS** res;

┌─res─┐

│ 2 │

└─────┘

##### Функции хэширования

Функции хэширования могут использоваться для детерминированного псевдослучайного разбрасывания элементов.

###### halfMD5

[Интерпретирует](https://clickhouse.tech/docs/ru/sql-reference/functions/hash-functions/#type_conversion_functions-reinterpretAsString) все входные параметры как строки и вычисляет хэш [MD5](https://ru.wikipedia.org/wiki/MD5) для каждой из них. Затем объединяет хэши, берет первые 8 байт хэша результирующей строки и интерпретирует их как значение типа UInt64 с big-endian порядком байтов.

Синтаксис:

halfMD5(par1, ...)

Функция относительно медленная (5 миллионов коротких строк в секунду на ядро процессора).

По возможности, используйте функцию [sipHash64](https://clickhouse.tech/docs/ru/sql-reference/functions/hash-functions/#hash_functions-siphash64) вместо неё.

Параметры:

* Функция принимает переменное число входных параметров. Параметры могут быть любого [поддерживаемого типа данных](https://clickhouse.tech/docs/ru/sql-reference/functions/hash-functions/).

Возвращаемое значение:

* Значение хэша с типом данных [UInt64](https://clickhouse.tech/docs/ru/sql-reference/functions/hash-functions/).

Пример:

**SELECT** halfMD5(array('e','x','a'), 'mple', 10, toDateTime('2019-06-15 23:00:00')) **AS** halfMD5hash, toTypeName(halfMD5hash) **AS** **type**

┌────────halfMD5hash─┬─type───┐

│ 186182704141653334 │ UInt64 │

└────────────────────┴────────┘

###### MD5

Вычисляет MD5 от строки и возвращает полученный набор байт в виде FixedString(16).

Если вам не нужен конкретно MD5, а нужен неплохой криптографический 128-битный хэш, то используйте вместо этого функцию sipHash128.

Если вы хотите получить такой же результат, как выдаёт утилита md5sum, напишите lower(hex(MD5(s))).

###### sipHash64

Генерирует 64-х битное значение [SipHash](https://131002.net/siphash/).

Синтаксис:

sipHash64(par1,...)

Это криптографическая хэш-функция. Она работает по крайней мере в три раза быстрее, чем функция [MD5](https://clickhouse.tech/docs/ru/sql-reference/functions/hash-functions/#hash_functions-md5).

Функция [интерпретирует](https://clickhouse.tech/docs/ru/sql-reference/functions/hash-functions/#type_conversion_functions-reinterpretAsString) все входные параметры как строки и вычисляет хэш MD5 для каждой из них. Затем комбинирует хэши по следующему алгоритму.

1. После хэширования всех входных параметров функция получает массив хэшей.
2. Функция принимает первый и второй элементы и вычисляет хэш для массива из них.
3. Затем функция принимает хэш-значение, вычисленное на предыдущем шаге, и третий элемент исходного хэш-массива, и вычисляет хэш для массива из них.
4. Предыдущий шаг повторяется для всех остальных элементов исходного хэш-массива.

Параметры:

* Функция принимает переменное число входных параметров. Параметры могут быть любого [поддерживаемого типа данных](https://clickhouse.tech/docs/ru/sql-reference/functions/hash-functions/).

Возвращаемое значение:

* Значение хэша с типом данных [UInt64](https://clickhouse.tech/docs/ru/sql-reference/functions/hash-functions/).

Пример:

**SELECT** sipHash64(array('e','x','a'), 'mple', 10, toDateTime('2019-06-15 23:00:00')) **AS** SipHash, toTypeName(SipHash) **AS** **type**

┌──────────────SipHash─┬─type───┐

│ 13726873534472839665 │ UInt64 │

└──────────────────────┴────────┘

###### sipHash128

Вычисляет SipHash от строки.

Принимает аргумент типа String. Возвращает FixedString(16).

Отличается от sipHash64 тем, что финальный xor-folding состояния делается только до 128 бит.

###### cityHash64

Генерирует 64-х битное значение [CityHash](https://github.com/google/cityhash).

Синтаксис:

cityHash64(par1,...)

Это не криптографическая хэш-функция. Она использует CityHash алгоритм для строковых параметров и зависящую от реализации быструю некриптографическую хэш-функцию для параметров с другими типами данных. Функция использует комбинатор CityHash для получения конечных результатов.

Параметры:

* Функция принимает переменное число входных параметров. Параметры могут быть любого [поддерживаемого типа данных](https://clickhouse.tech/docs/ru/sql-reference/functions/hash-functions/).

Возвращаемое значение:

* Значение хэша с типом данных [UInt64](https://clickhouse.tech/docs/ru/sql-reference/functions/hash-functions/).

Примеры:

Пример вызова:

**SELECT** cityHash64(array('e','x','a'), 'mple', 10, toDateTime('2019-06-15 23:00:00')) **AS** CityHash, toTypeName(CityHash) **AS** **type**

┌─────────────CityHash─┬─type───┐

│ 12072650598913549138 │ UInt64 │

└──────────────────────┴────────┘

А вот так вы можете вычислить чексумму всей таблицы с точностью до порядка строк:

**SELECT** groupBitXor(cityHash64(\*)) **FROM** **table**

###### intHash32

Вычисляет 32-битный хэш-код от целого числа любого типа.

Это сравнительно быстрая не криптографическая хэш-функция среднего качества для чисел.

###### intHash64

Вычисляет 64-битный хэш-код от целого числа любого типа.

Работает быстрее, чем intHash32. Качество среднее.

###### SHA1

###### SHA224

###### SHA256

Вычисляет SHA-1, SHA-224, SHA-256 от строки и возвращает полученный набор байт в виде FixedString(20), FixedString(28), FixedString(32).

Функция работает достаточно медленно (SHA-1 - примерно 5 миллионов коротких строк в секунду на одном процессорном ядре, SHA-224 и SHA-256 - примерно 2.2 миллионов).

Рекомендуется использовать эти функции лишь в тех случаях, когда вам нужна конкретная хэш-функция и вы не можете её выбрать.

Даже в этих случаях, рекомендуется применять функцию оффлайн - заранее вычисляя значения при вставке в таблицу, вместо того, чтобы применять её при SELECT-ах.

###### URLHash(url[, N])

Быстрая не криптографическая хэш-функция неплохого качества для строки, полученной из URL путём некоторой нормализации.

URLHash(s) – вычислить хэш от строки без одного завершающего символа /, ? или # на конце, если там такой есть.

URLHash(s, N) – вычислить хэш от строки до N-го уровня в иерархии URL, без одного завершающего символа /, ? или # на конце, если там такой есть.

Уровни аналогичные URLHierarchy.

###### farmFingerprint64

###### farmHash64

Создает 64-битное значение [FarmHash](https://github.com/google/farmhash), независимое от платформы (архитектуры сервера), что важно, если значения сохраняются или используются для разбиения данных на группы.

Синтаксис:

farmFingerprint64(par1, ...)

farmHash64(par1, ...)

Эти функции используют методы Fingerprint64 и Hash64 из всех [доступных методов](https://github.com/google/farmhash/blob/master/src/farmhash.h).

Параметры:

* Функция принимает переменное число входных параметров. Параметры могут быть любого [поддерживаемого типа данных](https://clickhouse.tech/docs/ru/sql-reference/functions/hash-functions/).

Возвращаемое значение:

* Значение хэша с типом данных [UInt64](https://clickhouse.tech/docs/ru/sql-reference/functions/hash-functions/).

Пример:

**SELECT** farmHash64(array('e','x','a'), 'mple', 10, toDateTime('2019-06-15 23:00:00')) **AS** FarmHash, toTypeName(FarmHash) **AS** **type**

┌─────────────FarmHash─┬─type───┐

│ 17790458267262532859 │ UInt64 │

└──────────────────────┴────────┘

###### javaHash

Вычисляет [JavaHash](http://hg.openjdk.java.net/jdk8u/jdk8u/jdk/file/478a4add975b/src/share/classes/java/lang/String.java#l1452) от строки. JavaHash не отличается ни скоростью, ни качеством, поэтому эту функцию следует считать устаревшей. Используйте эту функцию, если вам необходимо получить значение хэша по такому же алгоритму.

Синтаксис:

**SELECT** javaHash('');

Возвращаемое значение:

* Хэш-значение типа Int32.

Тип: javaHash.

Пример:

Запрос:

**SELECT** javaHash('Hello, world!');

Ответ:

┌─javaHash('Hello, world!')─┐

│ -1880044555 │

└───────────────────────────┘

###### javaHashUTF16LE

Вычисляет [JavaHash](http://hg.openjdk.java.net/jdk8u/jdk8u/jdk/file/478a4add975b/src/share/classes/java/lang/String.java#l1452) от строки, при допущении, что строка представлена в кодировке UTF-16LE.

Синтаксис:

javaHashUTF16LE(stringUtf16le)

Параметры:

* stringUtf16le – строка в UTF-16LE.

Возвращаемое значение:

* Хэш-значение типа Int32.

Тип: javaHash.

Пример:

Верный запрос для строки, кодированной в UTF-16LE.

Запрос:

**SELECT** javaHashUTF16LE(convertCharset('test', 'utf-8', 'utf-16le'))

Ответ:

┌─javaHashUTF16LE(convertCharset('test', 'utf-8', 'utf-16le'))─┐

│ 3556498 │

└──────────────────────────────────────────────────────────────┘

###### hiveHash

Вычисляет HiveHash от строки.

Синтаксис:

**SELECT** hiveHash('');

HiveHash – это результат [JavaHash](https://clickhouse.tech/docs/ru/sql-reference/functions/hash-functions/#hash_functions-javahash) с обнулённым битом знака числа. Функция используется в [Apache Hive](https://en.wikipedia.org/wiki/Apache_Hive) вплоть до версии 3.0.

Возвращаемое значение:

Хэш-значение типа Int32.

Тип: hiveHash.

Пример:

Запрос:

**SELECT** hiveHash('Hello, world!');

Ответ:

┌─hiveHash('Hello, world!')─┐

│ 267439093 │

└───────────────────────────┘

###### metroHash64

Генерирует 64-х битное значение [MetroHash](http://www.jandrewrogers.com/2015/05/27/metrohash/).

Синтаксис:

metroHash64(par1, ...)

Параметры

Функция принимает переменное число входных параметров. Параметры могут быть любого [поддерживаемого типа данных](https://clickhouse.tech/docs/ru/sql-reference/functions/hash-functions/).

Возвращаемое значение:

* Значение хэша с типом данных [UInt64](https://clickhouse.tech/docs/ru/sql-reference/functions/hash-functions/).

Пример:

**SELECT** metroHash64(array('e','x','a'), 'mple', 10, toDateTime('2019-06-15 23:00:00')) **AS** MetroHash, toTypeName(MetroHash) **AS** **type**

┌────────────MetroHash─┬─type───┐

│ 14235658766382344533 │ UInt64 │

└──────────────────────┴────────┘

###### jumpConsistentHash

Вычисляет JumpConsistentHash от значения типа UInt64.

Имеет два параметра: ключ типа UInt64 и количество бакетов. Возвращает значение типа Int32.

Дополнительные сведения смотрите по ссылке: [JumpConsistentHash](https://arxiv.org/pdf/1406.2294.pdf)

###### murmurHash2\_32, murmurHash2\_64

Генерирует значение [MurmurHash2](https://github.com/aappleby/smhasher).

Синтаксис:

murmurHash2\_32(par1, ...)

murmurHash2\_64(par1, ...)

Параметры:

Обе функции принимают переменное число входных параметров. Параметры могут быть любого [поддерживаемого типа данных](https://clickhouse.tech/docs/ru/sql-reference/functions/hash-functions/).

Возвращаемое значение:

* Функция murmurHash2\_32 возвращает значение типа [UInt32](https://clickhouse.tech/docs/ru/sql-reference/functions/hash-functions/).
* Функция murmurHash2\_64 возвращает значение типа [UInt64](https://clickhouse.tech/docs/ru/sql-reference/functions/hash-functions/).

Пример:

**SELECT** murmurHash2\_64(array('e','x','a'), 'mple', 10, toDateTime('2019-06-15 23:00:00')) **AS** MurmurHash2, toTypeName(MurmurHash2) **AS** **type**

┌──────────MurmurHash2─┬─type───┐

│ 11832096901709403633 │ UInt64 │

└──────────────────────┴────────┘

###### gccMurmurHash

Вычисляет 64-битное значение [MurmurHash2](https://github.com/aappleby/smhasher), используя те же hash seed, что и [gcc](https://github.com/gcc-mirror/gcc/blob/41d6b10e96a1de98e90a7c0378437c3255814b16/libstdc%2B%2B-v3/include/bits/functional_hash.h#L191).

Синтаксис:

gccMurmurHash(par1, ...);

Параметры:

* par1, ... – переменное число параметров. Каждый параметр может быть любого из [поддерживаемых типов данных](https://clickhouse.tech/docs/ru/sql-reference/data-types/).

Возвращаемое значение:

* Вычисленный хэш-код.

Тип: [UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Примеры:

Запрос:

**SELECT**

gccMurmurHash(1, 2, 3) **AS** res1,

gccMurmurHash(('a', [1, 2, 3], 4, (4, ['foo', 'bar'], 1, (1, 2)))) **AS** res2

Результат:

┌─────────────────res1─┬────────────────res2─┐

│ 12384823029245979431 │ 1188926775431157506 │

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

###### murmurHash3\_32, murmurHash3\_64

Генерирует значение [MurmurHash3](https://github.com/aappleby/smhasher).

Синтаксис:

murmurHash3\_32(par1, ...)

murmurHash3\_64(par1, ...)

Параметры:

Обе функции принимают переменное число входных параметров. Параметры могут быть любого [поддерживаемого типа данных](https://clickhouse.tech/docs/ru/sql-reference/functions/hash-functions/).

Возвращаемое значение:

* Функция murmurHash3\_32 возвращает значение типа [UInt32](https://clickhouse.tech/docs/ru/sql-reference/functions/hash-functions/).
* Функция murmurHash3\_64 возвращает значение типа [UInt64](https://clickhouse.tech/docs/ru/sql-reference/functions/hash-functions/).

Пример:

**SELECT** murmurHash3\_32(array('e','x','a'), 'mple', 10, toDateTime('2019-06-15 23:00:00')) **AS** MurmurHash3, toTypeName(MurmurHash3) **AS** **type**

┌─MurmurHash3─┬─type───┐

│ 2152717 │ UInt32 │

└─────────────┴────────┘

###### murmurHash3\_128

Генерирует значение [MurmurHash3](https://github.com/aappleby/smhasher).

Синтаксис:

murmurHash3\_128( expr )

Параметры:

* expr – [выражение](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-expressions) возвращающее значение типа[String](https://clickhouse.tech/docs/ru/sql-reference/functions/hash-functions/).

Возвращаемое значение:

* Хэш-значение типа [FixedString(16)](https://clickhouse.tech/docs/ru/sql-reference/functions/hash-functions/).

Пример:

**SELECT** murmurHash3\_128('example\_string') **AS** MurmurHash3, toTypeName(MurmurHash3) **AS** **type**

┌─MurmurHash3──────┬─type────────────┐

│ 6�1

�4"S5KT�~~q │ FixedString(16) │

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

###### xxHash32, xxHash64

Вычисляет xxHash от строки. Предлагается в двух вариантах: 32 и 64 бита.

**SELECT** xxHash32('');

**OR**

**SELECT** xxHash64('');

Возвращаемое значение:

* Хэш-значение типа Uint32 или Uint64.

Тип: xxHash.

Пример:

Запрос:

**SELECT** xxHash32('Hello, world!');

Ответ:

┌─xxHash32('Hello, world!')─┐

│ 834093149 │

└───────────────────────────┘

Смотрите также:

* [xxHash](https://cyan4973.github.io/xxHash/).

##### Функции генерации псевдослучайных чисел

Используются не криптографические генераторы псевдослучайных чисел.

Все функции принимают ноль аргументов или один аргумент.

В случае, если передан аргумент – он может быть любого типа, и его значение никак не используется.

Этот аргумент нужен только для того, чтобы предотвратить склейку одинаковых выражений – чтобы две разные записи одной функции возвращали разные столбцы, с разными случайными числами.

###### rand

Возвращает псевдослучайное число типа UInt32, равномерно распределённое среди всех чисел типа UInt32.

Используется linear congruential generator.

###### rand64

Возвращает псевдослучайное число типа UInt64, равномерно распределённое среди всех чисел типа UInt64.

Используется linear congruential generator.

###### randConstant

Создает константный столбец с псевдослучайным значением.

Синтаксис:

randConstant([x])

Параметры:

* x – [выражение](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-expressions), возвращающее значение одного из [поддерживаемых типов данных](https://clickhouse.tech/docs/ru/sql-reference/data-types/#data_types). Значение используется, чтобы избежать [склейки одинаковых выражений](https://clickhouse.tech/docs/ru/sql-reference/functions/#common-subexpression-elimination), если функция вызывается несколько раз в одном запросе. Необязательный параметр.

Возвращаемое значение:

* Псевдослучайное число.

Тип: [UInt32](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Пример:

Запрос:

**SELECT** rand(), rand(1), rand(number), randConstant(), randConstant(1), randConstant(number)

**FROM** numbers(3)

Результат:

┌─────rand()─┬────rand(1)─┬─rand(number)─┬─randConstant()─┬─randConstant(1)─┬─randConstant(number)─┐

│ 3047369878 │ 4132449925 │ 4044508545 │ 2740811946 │ 4229401477 │ 1924032898 │

│ 2938880146 │ 1267722397 │ 4154983056 │ 2740811946 │ 4229401477 │ 1924032898 │

│ 956619638 │ 4238287282 │ 1104342490 │ 2740811946 │ 4229401477 │ 1924032898 │

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

##### Случайные функции для работы со строками

###### randomString

###### randomFixedString

###### randomPrintableASCII

###### randomStringUTF8

###### fuzzBits

Синтаксис:

fuzzBits([s], [prob])

Инвертирует каждый бит s с вероятностью prob.

Параметры:

* s – String или FixedString.
* prob – constant Float32/64.

Возвращаемое значение:

Измененная случайным образом строка с тем же типом, что и s.

Пример:

Запрос:

**SELECT** fuzzBits(materialize('abacaba'), 0.1)

**FROM** numbers(3)

Результат:

┌─fuzzBits(materialize('abacaba'), 0.1)─┐

│ abaaaja │

│ a\*cjab+ │

│ aeca2A │

└───────────────────────────────────────┘

##### Функции кодирования

###### char

Возвращает строку, длина которой равна числу переданных аргументов, и каждый байт имеет значение соответствующего аргумента. Принимает несколько числовых аргументов. Если значение аргумента выходит за диапазон UInt8 (0..255), то оно преобразуется в UInt8 с возможным округлением и переполнением.

Синтаксис:

char(number\_1, [number\_2, ..., number\_n]);

Параметры:

* number\_1, number\_2, ..., number\_n – Числовые аргументы, которые интерпретируются как целые числа. Типы: [Int](https://clickhouse.tech/docs/ru/sql-reference/functions/encoding-functions/), [Float](https://clickhouse.tech/docs/ru/sql-reference/functions/encoding-functions/).

Возвращаемое значение:

* строка из соответствующих байт.

Тип: String.

Пример:

Запрос:

**SELECT** char(104.1, 101, 108.9, 108.9, 111) **AS** hello

Ответ:

┌─hello─┐

│ hello │

└───────┘

Вы можете создать строку в произвольной кодировке, передав соответствующие байты. Пример для UTF-8:

Запрос:

**SELECT** char(0xD0, 0xBF, 0xD1, 0x80, 0xD0, 0xB8, 0xD0, 0xB2, 0xD0, 0xB5, 0xD1, 0x82) **AS** hello;

Ответ:

┌─hello──┐

│ привет │

└────────┘

Запрос:

**SELECT** char(0xE4, 0xBD, 0xA0, 0xE5, 0xA5, 0xBD) **AS** hello;

Ответ:

┌─hello─┐

│ 你好 │

└───────┘

###### hex

Возвращает строку, содержащую шестнадцатеричное представление аргумента.

Синоним: HEX.

Синтаксис:

hex(arg)

Функция использует заглавные буквы A-F и не использует никаких префиксов (например, 0x) или суффиксов (например, h).

Для целочисленных аргументов он печатает шестнадцатеричные цифры («nibbles») от наиболее значимого до наименее значимого (big endian или «читаемый человеком» порядок). Он начинается с самого значимого ненулевого байта (начальные нулевые байты опущены), но всегда печатает обе цифры каждого байта, даже если начальная цифра равна нулю.

Пример:

Запрос:

**SELECT** hex(1);

Результат:

01

Значения типов Date and DateTime форматируются как соответствующие целые числа (количество дней с момента эпохи для Даты и значение временной метки Unix для даты и времени).

Для String and FixedString, все байты просто кодируются как два шестнадцатеричных числа. Нулевые байты не пропущены.

Значения типов с плавающей запятой и десятичной дробью кодируются как их представление в памяти. Поскольку мы поддерживаем архитектуру little endian, они кодируются в little endian. Нулевые начальные/конечные байты не опущены.

Параметры:

* arg – значение для преобразования в шестнадцатеричное. Типы: [String](https://clickhouse.tech/docs/ru/sql-reference/functions/encoding-functions/), [UInt](https://clickhouse.tech/docs/ru/sql-reference/functions/encoding-functions/), [Float](https://clickhouse.tech/docs/ru/sql-reference/functions/encoding-functions/), [Decimal](https://clickhouse.tech/docs/ru/sql-reference/functions/encoding-functions/), [Date](https://clickhouse.tech/docs/ru/sql-reference/functions/encoding-functions/) or [DateTime](https://clickhouse.tech/docs/ru/sql-reference/functions/encoding-functions/).

Возвращаемое значение:

* Строка с шестнадцатеричным представлением аргумента.

Type: String.

Пример:

Запрос:

**SELECT** hex(toFloat32(number)) **as** hex\_presentation **FROM** numbers(15, 2);

Результат:

┌─hex\_presentation─┐

│ 00007041 │

│ 00008041 │

└──────────────────┘

Query:

**SELECT** hex(toFloat64(number)) **as** hex\_presentation **FROM** numbers(15, 2);

Result:

┌─hex\_presentation─┐

│ 0000000000002E40 │

│ 0000000000003040 │

└──────────────────┘

###### unhex(str)

Принимает строку, содержащую любое количество шестнадцатеричных цифр, и возвращает строку, содержащую соответствующие байты. Поддерживает как прописные, так и строчные буквы A-F. Число шестнадцатеричных цифр не обязательно должно быть четным. Если он нечетный, то последняя цифра интерпретируется как наименее значимая половина байта 00-0F. Если строка аргумента содержит что-либо, кроме шестнадцатеричных цифр, возвращается некоторый результат, определенный реализацией (исключение не генерируется).

Если вы хотите преобразовать результат в число, вы можете использовать функции "reverse" и ‘reinterpretAsType’.

###### UUIDStringToNum(str)

Принимает строку, содержащую 36 символов в формате 123e4567-e89b-12d3-a456-426655440000, и возвращает в виде набора байт в FixedString(16).

###### UUIDNumToString(str)

Принимает значение типа FixedString(16). Возвращает строку из 36 символов в текстовом виде.

###### bitmaskToList(num)

Принимает целое число. Возвращает строку, содержащую список степеней двойки, в сумме дающих исходное число; по возрастанию, в текстовом виде, через запятую, без пробелов.

###### bitmaskToArray(num)

Принимает целое число. Возвращает массив чисел типа UInt64, содержащий степени двойки, в сумме дающих исходное число; числа в массиве идут по возрастанию.

##### Функции для работы с UUID

###### generateUUIDv4

Генерирует идентификатор [UUID версии 4](https://tools.ietf.org/html/rfc4122#section-4.4).

Синтаксис:

generateUUIDv4()

Возвращаемое значение:

* Значение типа [UUID](https://clickhouse.tech/docs/ru/sql-reference/functions/uuid-functions/).

Пример использования:

Этот пример демонстрирует, как создать таблицу с UUID-колонкой и добавить в нее сгенерированный UUID.

**CREATE** **TABLE** t\_uuid (x UUID) ENGINE=TinyLog

**INSERT** **INTO** t\_uuid **SELECT** generateUUIDv4()

**SELECT** \* **FROM** t\_uuid

┌────────────────────────────────────x─┐

│ f4bf890f-f9dc-4332-ad5c-0c18e73f28e9 │

└──────────────────────────────────────┘

###### toUUID (x)

Преобразует значение типа String в тип UUID.

Синтаксис:

toUUID(String)

Возвращаемое значение:

* Значение типа UUID.

Пример использования:

**SELECT** toUUID('61f0c404-5cb3-11e7-907b-a6006ad3dba0') **AS** uuid

┌─────────────────────────────────uuid─┐

│ 61f0c404-5cb3-11e7-907b-a6006ad3dba0 │

└──────────────────────────────────────┘

###### toUUIDOrNull (x)

Принимает строку, и пытается преобразовать в тип UUID. При неудаче возвращает NULL.

Синтаксис:

toUUIDOrNull(String)

Возвращаемое значение:

* Значение типа Nullable(UUID).

Пример использования:

**SELECT** toUUIDOrNull('61f0c404-5cb3-11e7-907b-a6006ad3dba0T') **AS** uuid

┌─uuid─┐

│ ᴺᵁᴸᴸ │

└──────┘

###### toUUIDOrZero (x)

Принимает строку, и пытается преобразовать в тип UUID. При неудаче возвращает нулевой UUID.

Синтаксис:

toUUIDOrZero(String)

Возвращаемое значение:

* Значение типа UUID.

Пример использования

**SELECT** toUUIDOrZero('61f0c404-5cb3-11e7-907b-a6006ad3dba0T') **AS** uuid

┌─────────────────────────────────uuid─┐

│ 00000000-0000-0000-0000-000000000000 │

└──────────────────────────────────────┘

###### UUIDStringToNum

Принимает строку, содержащую 36 символов в формате xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx, и возвращает в виде набора байт в [FixedString(16)](https://clickhouse.tech/docs/ru/sql-reference/functions/uuid-functions/).

Синтаксис:

UUIDStringToNum(String)

Возвращаемое значение:

FixedString(16)

Пример использования:

**SELECT**

'612f3c40-5d3b-217e-707b-6a546a3d7b29' **AS** uuid,

UUIDStringToNum(uuid) **AS** bytes

┌─uuid─────────────────────────────────┬─bytes────────────┐

│ 612f3c40-5d3b-217e-707b-6a546a3d7b29 │ a/<@];!~p{jTj={) │

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

###### UUIDNumToString

Принимает значение типа [FixedString(16)](https://clickhouse.tech/docs/ru/sql-reference/functions/uuid-functions/). Возвращает строку из 36 символов в текстовом виде.

Синтаксис:

UUIDNumToString(FixedString(16))

Возвращаемое значение:

* Значение типа String.

Пример использования:

**SELECT**

'a/<@];!~p{jTj={)' **AS** bytes,

UUIDNumToString(toFixedString(bytes, 16)) **AS** uuid

┌─bytes────────────┬─uuid─────────────────────────────────┐

│ a/<@];!~p{jTj={) │ 612f3c40-5d3b-217e-707b-6a546a3d7b29 │

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

См. также:

* [dictGetUUID](https://clickhouse.tech/docs/ru/sql-reference/functions/ext-dict-functions/).
* [dictGetUUIDOrDefault](https://clickhouse.tech/docs/ru/sql-reference/functions/ext-dict-functions/).

##### Функции для работы с URL

Все функции работают не по RFC - то есть, максимально упрощены ради производительности.

###### Функции, извлекающие часть URL-а

Если в URL-е нет ничего похожего, то возвращается пустая строка.

protocol:

* Возвращает протокол. Примеры: http, ftp, mailto, magnet…

domain:

* Извлекает имя хоста из URL.

Синтаксис:

**domain**(url)

Параметры:

* url – URL. Тип – [String](https://clickhouse.tech/docs/ru/sql-reference/functions/url-functions/).

URL может быть указан со схемой или без неё. Примеры:

svn+ssh://some.svn-hosting.com:80/repo/trunk

some.svn-hosting.com:80/repo/trunk

Для указанных примеров функция domain возвращает следующие результаты:

some.svn-hosting.com

some.svn-hosting.com

Возвращаемые значения:

* Имя хоста. Если RT.WideStore может распарсить входную строку как URL.
* Пустая строка. Если RT.WideStore не может распарсить входную строку как URL.

Тип – String.

Пример:

**SELECT** **domain**('svn+ssh://some.svn-hosting.com:80/repo/trunk')

┌─domain('svn+ssh://some.svn-hosting.com:80/repo/trunk')─┐

│ some.svn-hosting.com │

└────────────────────────────────────────────────────────┘

domainWithoutWWW

Возвращает домен, удалив префикс ‘www.’, если он присутствовал.

topLevelDomain

Извлекает домен верхнего уровня из URL.

Синтаксис:

topLevelDomain(url)

Параметры:

* url – URL. Тип – [String](https://clickhouse.tech/docs/ru/sql-reference/functions/url-functions/).

URL может быть указан со схемой или без неё. Примеры:

svn+ssh://some.svn-hosting.com:80/repo/trunk

some.svn-hosting.com:80/repo/trunk

Возвращаемые значения:

* Имя домена. Если RT.WideStore может распарсить входную строку как URL.
* Пустая строка. Если RT.WideStore не может распарсить входную строку как URL.

Тип – String.

Пример:

**SELECT** topLevelDomain('svn+ssh://www.some.svn-hosting.com:80/repo/trunk')

┌─topLevelDomain('svn+ssh://www.some.svn-hosting.com:80/repo/trunk')─┐

│ com │

└────────────────────────────────────────────────────────────────────┘

firstSignificantSubdomain

Возвращает «первый существенный поддомен». Первый существенный поддомен – это домен второго уровня, если он не равен одному из com, net, org, co, или домен третьего уровня, иначе. Список «несущественных» доменов второго уровня и другие детали реализации могут изменяться в будущем.

cutToFirstSignificantSubdomain

Возвращает часть домена, включающую поддомены верхнего уровня до «первого существенного поддомена» (см. выше).

cutToFirstSignificantSubdomainCustom

Возвращает часть домена, включающую поддомены верхнего уровня до первого существенного поддомена. Принимает имя пользовательского [списка доменов верхнего уровня](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_%D0%B4%D0%BE%D0%BC%D0%B5%D0%BD%D0%BE%D0%B2_%D0%B2%D0%B5%D1%80%D1%85%D0%BD%D0%B5%D0%B3%D0%BE_%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D1%8F).

Полезно, если требуется актуальный список доменов верхнего уровня или если есть пользовательский.

Пример конфигурации:

<!-- <top\_level\_domains\_path>/var/lib/clickhouse/top\_level\_domains/</top\_level\_domains\_path> -->

<top\_level\_domains\_lists>

<!-- https://publicsuffix.org/list/public\_suffix\_list.dat -->

<public\_suffix\_list>public\_suffix\_list.dat</public\_suffix\_list>

<!-- NOTE: path is under top\_level\_domains\_path -->

</top\_level\_domains\_lists>

Синтаксис:

cutToFirstSignificantSubdomain(URL, TLD)

Параметры:

* URL – URL. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).
* TLD – имя пользовательского списка доменов верхнего уровня. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).

Возвращаемое значение:

* Часть домена, включающая поддомены верхнего уровня до первого существенного поддомена.

Тип: [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).

Пример:

Запрос:

**SELECT** cutToFirstSignificantSubdomainCustom('bar.foo.there-is-no-such-domain', 'public\_suffix\_list');

Результат:

┌─cutToFirstSignificantSubdomainCustom('bar.foo.there-is-no-such-domain', 'public\_suffix\_list')─┐

│ foo.there-is-no-such-domain │

└───────────────────────────────────────────────────────────────────────────────────────────────┘

Смотрите также:

* [firstSignificantSubdomain](https://clickhouse.tech/docs/ru/sql-reference/functions/url-functions/#firstsignificantsubdomain).

cutToFirstSignificantSubdomainCustomWithWWW

Возвращает часть домена, включающую поддомены верхнего уровня до первого существенного поддомена, не опуская "www". Принимает имя пользовательского списка доменов верхнего уровня.

Полезно, если требуется актуальный список доменов верхнего уровня или если есть пользовательский.

Пример конфигурации:

<!-- <top\_level\_domains\_path>/var/lib/clickhouse/top\_level\_domains/</top\_level\_domains\_path> -->

<top\_level\_domains\_lists>

<!-- https://publicsuffix.org/list/public\_suffix\_list.dat -->

<public\_suffix\_list>public\_suffix\_list.dat</public\_suffix\_list>

<!-- NOTE: path is under top\_level\_domains\_path -->

</top\_level\_domains\_lists>

Синтаксис:

cutToFirstSignificantSubdomainCustomWithWWW(URL, TLD)

Параметры:

* URL – URL. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).
* TLD – имя пользовательского списка доменов верхнего уровня. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).

Возвращаемое значение:

* Часть домена, включающая поддомены верхнего уровня до первого существенного поддомена, без удаления www.

Тип: [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).

Пример:

Запрос:

**SELECT** cutToFirstSignificantSubdomainCustomWithWWW('www.foo', 'public\_suffix\_list');

Результат:

┌─cutToFirstSignificantSubdomainCustomWithWWW('www.foo', 'public\_suffix\_list')─┐

│ www.foo │

└──────────────────────────────────────────────────────────────────────────────┘

Смотрите также:

* [firstSignificantSubdomain](https://clickhouse.tech/docs/ru/sql-reference/functions/url-functions/#firstsignificantsubdomain).

firstSignificantSubdomainCustom

Возвращает первый существенный поддомен. Принимает имя пользовательского списка доменов верхнего уровня.

Полезно, если требуется актуальный список доменов верхнего уровня или если есть пользовательский.

Пример конфигурации:

<!-- <top\_level\_domains\_path>/var/lib/clickhouse/top\_level\_domains/</top\_level\_domains\_path> -->

<top\_level\_domains\_lists>

<!-- https://publicsuffix.org/list/public\_suffix\_list.dat -->

<public\_suffix\_list>public\_suffix\_list.dat</public\_suffix\_list>

<!-- NOTE: path is under top\_level\_domains\_path -->

</top\_level\_domains\_lists>

Синтаксис:

firstSignificantSubdomainCustom(URL, TLD)

Параметры:

* URL – URL. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).
* TLD – имя пользовательского списка доменов верхнего уровня. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).

Возвращаемое значение:

* Первый существенный поддомен.

Тип: [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).

Пример:

Запрос:

**SELECT** firstSignificantSubdomainCustom('bar.foo.there-is-no-such-domain', 'public\_suffix\_list');

Результат:

┌─firstSignificantSubdomainCustom('bar.foo.there-is-no-such-domain', 'public\_suffix\_list')─┐

│ foo │

└──────────────────────────────────────────────────────────────────────────────────────────┘

Смотрите также:

* [firstSignificantSubdomain](https://clickhouse.tech/docs/ru/sql-reference/functions/url-functions/#firstsignificantsubdomain).

port(URL[, default\_port = 0])

Возвращает порт или значение default\_port, если в URL-адресе нет порта (или передан невалидный URL)

path

Возвращает путь. Пример: /top/news.html Путь не включает в себя query string.

pathFull

То же самое, но включая query string и fragment. Пример: /top/news.html?page=2#comments

queryString

Возвращает query-string. Пример: page=1&lr=213. query-string не включает в себя начальный знак вопроса, а также # и всё, что после #.

fragment

Возвращает fragment identifier. fragment не включает в себя начальный символ решётки.

queryStringAndFragment

Возвращает query string и fragment identifier. Пример: страница=1#29390.

extractURLParameter(URL, name)

Возвращает значение параметра name в URL, если такой есть; или пустую строку, иначе; если параметров с таким именем много - вернуть первый попавшийся. Функция работает при допущении, что имя параметра закодировано в URL в точности таким же образом, что и в переданном аргументе.

extractURLParameters(URL)

Возвращает массив строк вида name=value, соответствующих параметрам URL. Значения никак не декодируются.

extractURLParameterNames(URL)

Возвращает массив строк вида name, соответствующих именам параметров URL. Значения никак не декодируются.

URLHierarchy(URL)

Возвращает массив, содержащий URL, обрезанный с конца по символам /, ? в пути и query-string. Подряд идущие символы-разделители считаются за один. Резка производится в позиции после всех подряд идущих символов-разделителей.

URLPathHierarchy(URL)

То же самое, но без протокола и хоста в результате. Элемент / (корень) не включается. Пример: Функция используется для реализации древовидных отчётов по URL.

URLPathHierarchy('https://example.com/browse/CONV-6788') =

[

'/browse/',

'/browse/CONV-6788'

]

decodeURLComponent(URL)

Возвращает декодированный URL.

Пример:

**SELECT** decodeURLComponent('http://127.0.0.1:8123/?query=SELECT%201%3B') **AS** DecodedURL;

┌─DecodedURL─────────────────────────────┐

│ http://127.0.0.1:8123/?query=SELECT 1; │

└────────────────────────────────────────┘

netloc

Извлекает сетевую локальность (username:password@host:port) из URL.

Синтаксис:

netloc(URL)

Параметры:

* url – URL. Тип – [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).

Возвращаемое значение:

* username:password@host:port.

Тип: String.

Пример:

Запрос:

**SELECT** netloc('http://paul@www.example.com:80/');

Результат:

┌─netloc('http://paul@www.example.com:80/')─┐

│ paul@www.example.com:80 │

└───────────────────────────────────────────┘

###### Функции, удаляющие часть из URL-а

Если в URL-е нет ничего похожего, то URL остаётся без изменений.

cutWWW

Удаляет не более одного ‘www.’ с начала домена URL-а, если есть.

cutQueryString

Удаляет query string. Знак вопроса тоже удаляется.

cutFragment

Удаляет fragment identifier. Символ решётки тоже удаляется.

cutQueryStringAndFragment

Удаляет query string и fragment identifier. Знак вопроса и символ решётки тоже удаляются.

cutURLParameter(URL, name)

Удаляет параметр URL с именем name, если такой есть. Функция работает при допущении, что имя параметра закодировано в URL в точности таким же образом, что и в переданном аргументе.

##### Функции для работы с IP-адресами

###### IPv4NumToString(num)

Принимает число типа UInt32. Интерпретирует его, как IPv4-адрес в big endian. Возвращает строку, содержащую соответствующий IPv4-адрес в формате A.B.C.D (числа в десятичной форме через точки).

Синоним: INET\_NTOA.

###### IPv4StringToNum(s)

Функция, обратная к IPv4NumToString. Если IPv4 адрес в неправильном формате, то возвращает 0.

Синоним: INET\_ATON.

###### IPv4NumToStringClassC(num)

Похоже на IPv4NumToString, но вместо последнего октета используется xxx.

Пример:

**SELECT**

IPv4NumToStringClassC(ClientIP) **AS** k,

**count**() **AS** **c**

**FROM** test.hits

**GROUP** **BY** k

**ORDER** **BY** **c** **DESC**

**LIMIT** 10

┌─k──────────────┬─────c─┐

│ 83.149.9.xxx │ 26238 │

│ 217.118.81.xxx │ 26074 │

│ 213.87.129.xxx │ 25481 │

│ 83.149.8.xxx │ 24984 │

│ 217.118.83.xxx │ 22797 │

│ 78.25.120.xxx │ 22354 │

│ 213.87.131.xxx │ 21285 │

│ 78.25.121.xxx │ 20887 │

│ 188.162.65.xxx │ 19694 │

│ 83.149.48.xxx │ 17406 │

└────────────────┴───────┘

В связи с тем, что использование xxx весьма необычно, это может быть изменено в дальнейшем. Вам не следует полагаться на конкретный вид этого фрагмента.

IPv6NumToString(x)

Принимает значение типа FixedString(16), содержащее IPv6-адрес в бинарном виде. Возвращает строку, содержащую этот адрес в текстовом виде.

IPv6-mapped IPv4 адреса выводится в формате ::ffff:111.222.33.44.

Примеры: INET6\_NTOA.

Примеры:

**SELECT** IPv6NumToString(toFixedString(unhex('2A0206B8000000000000000000000011'), 16)) **AS** addr

┌─addr─────────┐

│ 2a02:6b8::11 │

└──────────────┘

**SELECT**

IPv6NumToString(ClientIP6 **AS** k),

**count**() **AS** **c**

**FROM** hits\_all

**WHERE** EventDate = today() **AND** **substring**(ClientIP6, 1, 12) != unhex('00000000000000000000FFFF')

**GROUP** **BY** k

**ORDER** **BY** **c** **DESC**

**LIMIT** 10

┌─IPv6NumToString(ClientIP6)──────────────┬─────c─┐

│ 2a02:2168:aaa:bbbb::2 │ 24695 │

│ 2a02:2698:abcd:abcd:abcd:abcd:8888:5555 │ 22408 │

│ 2a02:6b8:0:fff::ff │ 16389 │

│ 2a01:4f8:111:6666::2 │ 16016 │

│ 2a02:2168:888:222::1 │ 15896 │

│ 2a01:7e00::ffff:ffff:ffff:222 │ 14774 │

│ 2a02:8109:eee:ee:eeee:eeee:eeee:eeee │ 14443 │

│ 2a02:810b:8888:888:8888:8888:8888:8888 │ 14345 │

│ 2a02:6b8:0:444:4444:4444:4444:4444 │ 14279 │

│ 2a01:7e00::ffff:ffff:ffff:ffff │ 13880 │

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

**SELECT**

IPv6NumToString(ClientIP6 **AS** k),

**count**() **AS** **c**

**FROM** hits\_all

**WHERE** EventDate = today()

**GROUP** **BY** k

**ORDER** **BY** **c** **DESC**

**LIMIT** 10

┌─IPv6NumToString(ClientIP6)─┬──────c─┐

│ ::ffff:94.26.111.111 │ 747440 │

│ ::ffff:37.143.222.4 │ 529483 │

│ ::ffff:5.166.111.99 │ 317707 │

│ ::ffff:46.38.11.77 │ 263086 │

│ ::ffff:79.105.111.111 │ 186611 │

│ ::ffff:93.92.111.88 │ 176773 │

│ ::ffff:84.53.111.33 │ 158709 │

│ ::ffff:217.118.11.22 │ 154004 │

│ ::ffff:217.118.11.33 │ 148449 │

│ ::ffff:217.118.11.44 │ 148243 │

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

###### IPv6StringToNum

Функция, обратная к [IPv6NumToString](https://clickhouse.tech/docs/ru/sql-reference/functions/ip-address-functions/#ipv6numtostringx). Если IPv6 адрес передан в неправильном формате, то возвращает строку из нулевых байт.

Если IP адрес является корректным IPv4 адресом, функция возвращает его IPv6 эквивалент.

HEX может быть в любом регистре.

Синоним: INET6\_ATON.

Синтаксис:

IPv6StringToNum(string)

Аргумент:

* string – IP адрес. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).

Возвращаемое значение:

* Адрес IPv6 в двоичном представлении.

Тип: [FixedString(16)](https://clickhouse.tech/docs/ru/sql-reference/data-types/fixedstring/).

Пример:

Запрос:

**SELECT** addr, cutIPv6(IPv6StringToNum(addr), 0, 0) **FROM** (**SELECT** ['notaddress', '127.0.0.1', '1111::ffff'] **AS** addr) ARRAY **JOIN** addr;

Результат:

┌─addr───────┬─cutIPv6(IPv6StringToNum(addr), 0, 0)─┐

│ notaddress │ :: │

│ 127.0.0.1 │ ::ffff:127.0.0.1 │

│ 1111::ffff │ 1111::ffff │

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

Смотрите также:

* [cutIPv6](https://clickhouse.tech/docs/ru/sql-reference/functions/ip-address-functions/#cutipv6x-bytestocutforipv6-bytestocutforipv4).

###### IPv4ToIPv6(x)

Принимает число типа UInt32. Интерпретирует его, как IPv4-адрес в [big endian](https://en.wikipedia.org/wiki/Endianness). Возвращает значение FixedString(16), содержащее адрес IPv6 в двоичном формате.

Пример:

**SELECT** IPv6NumToString(IPv4ToIPv6(IPv4StringToNum('192.168.0.1'))) **AS** addr

┌─addr───────────────┐

│ ::ffff:192.168.0.1 │

└────────────────────┘

###### cutIPv6(x, bytesToCutForIPv6, bytesToCutForIPv4)

Принимает значение типа FixedString(16), содержащее IPv6-адрес в бинарном виде. Возвращает строку, содержащую адрес из указанного количества байтов, удаленных в текстовом формате. Например:

**WITH**

IPv6StringToNum('2001:0DB8:AC10:FE01:FEED:BABE:CAFE:F00D') **AS** ipv6,

IPv4ToIPv6(IPv4StringToNum('192.168.0.1')) **AS** ipv4

**SELECT**

cutIPv6(ipv6, 2, 0),

cutIPv6(ipv4, 0, 2)

┌─cutIPv6(ipv6, 2, 0)─────────────────┬─cutIPv6(ipv4, 0, 2)─┐

│ 2001:db8:ac10:fe01:feed:babe:cafe:0 │ ::ffff:192.168.0.0 │

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

###### IPv4CIDRToRange(ipv4, Cidr)

Принимает на вход IPv4 и значение UInt8, содержащее [CIDR](https://ru.wikipedia.org/wiki/%D0%91%D0%B5%D1%81%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%BE%D0%B2%D0%B0%D1%8F_%D0%B0%D0%B4%D1%80%D0%B5%D1%81%D0%B0%D1%86%D0%B8%D1%8F). Возвращает кортеж с двумя IPv4, содержащими нижний и более высокий диапазон подсети.

**SELECT** IPv4CIDRToRange(toIPv4('192.168.5.2'), 16)

┌─IPv4CIDRToRange(toIPv4('192.168.5.2'), 16)─┐

│ ('192.168.0.0','192.168.255.255') │

└────────────────────────────────────────────┘

###### IPv6CIDRToRange(ipv6, Cidr)

Принимает на вход IPv6 и значение UInt8, содержащее CIDR. Возвращает кортеж с двумя IPv6, содержащими нижний и более высокий диапазон подсети.

**SELECT** IPv6CIDRToRange(toIPv6('2001:0db8:0000:85a3:0000:0000:ac1f:8001'), 32)

┌─IPv6CIDRToRange(toIPv6('2001:0db8:0000:85a3:0000:0000:ac1f:8001'), 32)─┐

│ ('2001:db8::','2001:db8:ffff:ffff:ffff:ffff:ffff:ffff') │

└────────────────────────────────────────────────────────────────────────┘

###### toIPv4(string)

Псевдоним функции IPv4StringToNum() которая принимает строку с адресом IPv4 и возвращает значение типа [IPv4](https://clickhouse.tech/docs/ru/sql-reference/functions/ip-address-functions/), которое равно значению, возвращаемому функцией IPv4StringToNum().

**WITH**

'171.225.130.45' **as** IPv4\_string

**SELECT**

toTypeName(IPv4StringToNum(IPv4\_string)),

toTypeName(toIPv4(IPv4\_string))

┌─toTypeName(IPv4StringToNum(IPv4\_string))─┬─toTypeName(toIPv4(IPv4\_string))─┐

│ UInt32 │ IPv4 │

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

**WITH**

'171.225.130.45' **as** IPv4\_string

**SELECT**

hex(IPv4StringToNum(IPv4\_string)),

hex(toIPv4(IPv4\_string))

┌─hex(IPv4StringToNum(IPv4\_string))─┬─hex(toIPv4(IPv4\_string))─┐

│ ABE1822D │ ABE1822D │

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

###### toIPv6

Приводит строку с адресом в формате IPv6 к типу [IPv6](https://clickhouse.tech/docs/ru/sql-reference/data-types/domains/ipv6/). Возвращает пустое значение, если входящая строка не является корректным IP адресом.

Похоже на функцию [IPv6StringToNum](https://clickhouse.tech/docs/ru/sql-reference/functions/ip-address-functions/#ipv6stringtonums), которая представляет адрес IPv6 в двоичном виде.

Если входящая строка содержит корректный IPv4 адрес, функция возвращает его IPv6 эквивалент.

Синтаксис:

toIPv6(string)

Аргумент:

* string – IP адрес. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)

Возвращаемое значение:

* IP адрес.

Тип: [IPv6](https://clickhouse.tech/docs/ru/sql-reference/data-types/domains/ipv6/).

Примеры:

Запрос:

**WITH** '2001:438:ffff::407d:1bc1' **AS** IPv6\_string

**SELECT**

hex(IPv6StringToNum(IPv6\_string)),

hex(toIPv6(IPv6\_string));

Результат:

┌─hex(IPv6StringToNum(IPv6\_string))─┬─hex(toIPv6(IPv6\_string))─────────┐

│ 20010438FFFF000000000000407D1BC1 │ 20010438FFFF000000000000407D1BC1 │

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

Запрос:

**SELECT** toIPv6('127.0.0.1');

Результат:

┌─toIPv6('127.0.0.1')─┐

│ ::ffff:127.0.0.1 │

└─────────────────────┘

###### isIPv4String

Определяет, является ли строка адресом IPv4 или нет. Также вернет 0, если string – адрес IPv6.

Синтаксис:

isIPv4String(string)

Параметры:

* string – IP адрес. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).

Возвращаемое значение:

* 1 если string является адресом IPv4 , иначе – 0.

Тип: [UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Примеры:

Запрос:

**SELECT** addr, isIPv4String(addr) **FROM** ( **SELECT** ['0.0.0.0', '127.0.0.1', '::ffff:127.0.0.1'] **AS** addr ) ARRAY **JOIN** addr

Результат:

┌─addr─────────────┬─isIPv4String(addr)─┐

│ 0.0.0.0 │ 1 │

│ 127.0.0.1 │ 1 │

│ ::ffff:127.0.0.1 │ 0 │

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

###### isIPv6String

Определяет, является ли строка адресом IPv6 или нет. Также вернет 0, если string – адрес IPv4.

Синтаксис:

isIPv6String(string)

Параметры:

* string – IP адрес. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).

Возвращаемое значение:

* 1 если string является адресом IPv6, иначе – 0.

Тип: [UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Примеры:

Запрос:

**SELECT** addr, isIPv6String(addr) **FROM** ( **SELECT** ['::', '1111::ffff', '::ffff:127.0.0.1', '127.0.0.1'] **AS** addr ) ARRAY **JOIN** addr

Результат:

┌─addr─────────────┬─isIPv6String(addr)─┐

│ :: │ 1 │

│ 1111::ffff │ 1 │

│ ::ffff:127.0.0.1 │ 1 │

│ 127.0.0.1 │ 0 │

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

##### Функции для работы с внешними словарями

Информацию о подключении и настройке внешних словарей смотрите в разделе [Внешние словари](https://clickhouse.tech/docs/ru/sql-reference/dictionaries/external-dictionaries/external-dicts/).

###### dictGet

Извлекает значение из внешнего словаря.

Синтаксис:

dictGet('dict\_name', 'attr\_name', id\_expr)

dictGetOrDefault('dict\_name', 'attr\_name', id\_expr, default\_value\_expr)

Параметры:

* dict\_name – имя словаря. [Строковый литерал](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-string-literal).
* attr\_name – имя столбца словаря. [Строковый литерал](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-string-literal).
* id\_expr – значение ключа словаря. [Выражение](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-expressions), возвращающее значение типа [UInt64](https://clickhouse.tech/docs/ru/sql-reference/functions/ext-dict-functions/) или [Tuple](https://clickhouse.tech/docs/ru/sql-reference/functions/ext-dict-functions/) в зависимости от конфигурации словаря.
* default\_value\_expr – значение, возвращаемое в том случае, когда словарь не содержит строки с заданным ключом id\_expr. [Выражение](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-expressions) возвращающее значение с типом данных, сконфигурированным для атрибута attr\_name.

Возвращаемое значение:

* Значение атрибута, соответствующее ключу id\_expr, если RT.WideStore смог привести это значение к [заданному типу данных](https://clickhouse.tech/docs/ru/sql-reference/functions/ext-dict-functions/#ext_dict_structure-attributes).
* Если ключа, соответствующего id\_expr в словаре нет, то:
  + dictGet возвращает содержимое элемента <null\_value>, указанного для атрибута в конфигурации словаря.
  + dictGetOrDefault возвращает атрибут default\_value\_expr.

Если значение атрибута не удалось обработать или оно не соответствует типу данных атрибута, то RT.WideStore генерирует исключение.

Пример:

Создадим текстовый файл ext-dict-text.csv со следующим содержимым:

1,1

2,2

Первый столбец – id, второй столбец – c1.

Настройка внешнего словаря:

<widestore>

<dictionary>

<name>ext-dict-test</name>

<source>

<file>

<path>/path-to/ext-dict-test.csv</path>

<format>CSV</format>

</file>

</source>

<layout>

<flat />

</layout>

<structure>

<id>

<name>id</name>

</id>

<attribute>

<name>c1</name>

<type>UInt32</type>

<null\_value></null\_value>

</attribute>

</structure>

<lifetime>0</lifetime>

</dictionary>

</widestore>

Выполним запрос:

**SELECT**

dictGetOrDefault('ext-dict-test', 'c1', number + 1, toUInt32(number \* 10)) **AS** val,

toTypeName(val) **AS** **type**

**FROM** **system**.numbers

**LIMIT** 3

┌─val─┬─type───┐

│ 1 │ UInt32 │

│ 2 │ UInt32 │

│ 20 │ UInt32 │

└─────┴────────┘

Смотрите также:

* [Внешние словари](https://clickhouse.tech/docs/ru/sql-reference/functions/ext-dict-functions/).

###### dictHas

Проверяет, присутствует ли запись с указанным ключом в словаре.

Синтаксис:

dictHas('dict\_name', id)

Параметры:

* dict\_name – имя словаря. [Строковый литерал](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-string-literal).
* id\_expr – значение ключа словаря. [Выражение](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-expressions), возвращающее значение типа [UInt64](https://clickhouse.tech/docs/ru/sql-reference/functions/ext-dict-functions/) или [Tuple](https://clickhouse.tech/docs/ru/sql-reference/functions/ext-dict-functions/) в зависимости от конфигурации словаря.

Возвращаемое значение:

* 0, если ключа нет.
* 1, если ключ есть.

Тип – UInt8.

###### dictGetHierarchy

Создаёт массив, содержащий цепочку предков для заданного ключа в [иерархическом словаре](https://clickhouse.tech/docs/ru/sql-reference/dictionaries/external-dictionaries/external-dicts-dict-hierarchical/).

Синтаксис:

dictGetHierarchy('dict\_name', **key**)

Параметры:

* dict\_name – имя словаря. [Строковый литерал](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-string-literal).
* key – значение ключа. [Выражение](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-expressions), возвращающее значение типа [UInt64](https://clickhouse.tech/docs/ru/sql-reference/functions/ext-dict-functions/).

Возвращаемое значение:

* Цепочка предков заданного ключа.

Type: [Array(UInt64)](https://clickhouse.tech/docs/ru/sql-reference/functions/ext-dict-functions/).

###### dictIsIn

Проверяет предка ключа по всей иерархической цепочке словаря.

dictIsIn ('dict\_name', child\_id\_expr, ancestor\_id\_expr)

Параметры:

* dict\_name – имя словаря. [Строковый литерал](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-string-literal).
* child\_id\_expr – ключ для проверки. [Выражение](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-expressions), возвращающее значение типа [UInt64](https://clickhouse.tech/docs/ru/sql-reference/functions/ext-dict-functions/).
* ancestor\_id\_expr – предполагаемый предок ключа child\_id\_expr. [Выражение](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-expressions), возвращающее значение типа [UInt64](https://clickhouse.tech/docs/ru/sql-reference/functions/ext-dict-functions/).

Возвращаемое значение:

* 0, если child\_id\_expr – не дочерний элемент ancestor\_id\_expr.
* 1, если child\_id\_expr – дочерний элемент ancestor\_id\_expr или если child\_id\_expr и есть ancestor\_id\_expr.

Тип – UInt8.

###### Прочие функции

RT.WideStore поддерживает специализированные функции, которые приводят значения атрибутов словаря к определённому типу данных независимо от конфигурации словаря.

Функции:

* dictGetInt8, dictGetInt16, dictGetInt32, dictGetInt64;
* dictGetUInt8, dictGetUInt16, dictGetUInt32, dictGetUInt64;
* dictGetFloat32, dictGetFloat64;
* dictGetDate;
* dictGetDateTime;
* dictGetUUID;
* dictGetString.

Все эти функции можно использовать с модификатором OrDefault. Например, dictGetDateOrDefault.

Синтаксис:

dictGet[**Type**]('dict\_name', 'attr\_name', id\_expr)

dictGet[**Type**]OrDefault('dict\_name', 'attr\_name', id\_expr, default\_value\_expr)

Параметры:

* dict\_name – имя словаря. [Строковый литерал](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-string-literal).
* attr\_name – имя столбца словаря. [Строковый литерал](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-string-literal).
* id\_expr – значение ключа словаря. [Выражение](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-expressions), возвращающее значение типа [UInt64](https://clickhouse.tech/docs/ru/sql-reference/functions/ext-dict-functions/) или [Tuple](https://clickhouse.tech/docs/ru/sql-reference/functions/ext-dict-functions/) в зависимости от конфигурации словаря.
* default\_value\_expr – значение, возвращаемое в том случае, когда словарь не содержит строки с заданным ключом id\_expr. [Выражение](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-expressions) возвращающее значение с типом данных, сконфигурированным для атрибута attr\_name.

Возвращаемое значение:

* Если RT.WideStore успешно обработал атрибут в соответствии с [заданным типом данных](https://clickhouse.tech/docs/ru/sql-reference/functions/ext-dict-functions/#ext_dict_structure-attributes), то функции возвращают значение атрибута, соответствующее ключу id\_expr.
* Если запрошенного id\_expr нет в словаре, то:
  + dictGet[Type] возвращает содержимое элемента <null\_value>, указанного для атрибута в конфигурации словаря.
  + dictGet[Type]OrDefault возвращает аргумент default\_value\_expr.

Если значение атрибута не удалось обработать или оно не соответствует типу данных атрибута, то RT.WideStore генерирует исключение.

##### Функции для работы со словарями

Чтобы указанные ниже функции работали, в конфиге сервера должны быть указаны пути и адреса для получения всех словарей. Словари загружаются при первом вызове любой из этих функций. Если справочники не удаётся загрузить - будет выкинуто исключение.

О том, как создать справочники, смотрите в разделе «Словари».

###### Множественные геобазы

RT.WideStore поддерживает работу одновременно с несколькими альтернативными геобазами (иерархиями регионов), для того чтобы можно было поддержать разные точки зрения о принадлежности регионов странам.

В конфиге clickhouse-server указывается файл с иерархией регионов:

<path\_to\_regions\_hierarchy\_file>/opt/geo/regions\_hierarchy.txt</path\_to\_regions\_hierarchy\_file>

Кроме указанного файла, рядом ищутся файлы, к имени которых (до расширения) добавлен символ \_ и какой угодно суффикс.

Например, также найдётся файл /opt/geo/regions\_hierarchy\_ua.txt, если такой есть.

ua называется ключом словаря. Для словаря без суффикса, ключ является пустой строкой.

Все словари перезагружаются в рантайме (раз в количество секунд, заданное в конфигурационном параметре builtin\_dictionaries\_reload\_interval, по умолчанию - раз в час), но перечень доступных словарей определяется один раз, при старте сервера.

Во все функции по работе с регионами, в конце добавлен один необязательный аргумент – ключ словаря. Далее он обозначен как geobase.

Пример:

regionToCountry(RegionID) - использует словарь по умолчанию: /opt/geo/regions\_hierarchy.txt;

regionToCountry(RegionID, '') - использует словарь по умолчанию: /opt/geo/regions\_hierarchy.txt;

regionToCountry(RegionID, 'ua') - использует словарь для ключа ua: /opt/geo/regions\_hierarchy\_ua.txt;

regionToCity(id[, geobase])

Принимает число типа UInt32 – идентификатор региона из геобазы. Если регион является городом или входит в некоторый город, то возвращает идентификатор региона – соответствующего города. Иначе возвращает 0.

regionToArea(id[, geobase])

Переводит регион в область (тип в геобазе – 5). В остальном, аналогично функции regionToCity.

**SELECT** **DISTINCT** regionToName(regionToArea(toUInt32(number), 'ua'))

**FROM** **system**.numbers

**LIMIT** 15

┌─regionToName(regionToArea(toUInt32(number), \'ua\'))─┐

│ │

│ Москва и Московская область │

│ Санкт-Петербург и Ленинградская область │

│ Белгородская область │

│ Ивановская область │

│ Калужская область │

│ Костромская область │

│ Курская область │

│ Липецкая область │

│ Орловская область │

│ Рязанская область │

│ Смоленская область │

│ Тамбовская область │

│ Тверская область │

│ Тульская область │

└──────────────────────────────────────────────────────┘

regionToDistrict(id[, geobase])

Переводит регион в федеральный округ (тип в геобазе – 4). В остальном, аналогично функции regionToCity.

**SELECT** **DISTINCT** regionToName(regionToDistrict(toUInt32(number), 'ua'))

**FROM** **system**.numbers

**LIMIT** 15

┌─regionToName(regionToDistrict(toUInt32(number), \'ua\'))─┐

│ │

│ Центральный федеральный округ │

│ Северо-Западный федеральный округ │

│ Южный федеральный округ │

│ Северо-Кавказский федеральный округ │

│ Приволжский федеральный округ │

│ Уральский федеральный округ │

│ Сибирский федеральный округ │

│ Дальневосточный федеральный округ │

│ Шотландия │

│ Фарерские острова │

│ Фламандский регион │

│ Брюссельский столичный регион │

│ Валлония │

│ Федерация Боснии и Герцеговины │

└──────────────────────────────────────────────────────────┘

regionToCountry(id[, geobase])

Переводит регион в страну. В остальном, аналогично функции regionToCity.

Пример: regionToCountry(toUInt32(213)) = 225 – преобразовали Москву (213) в Россию (225).

regionToContinent(id[, geobase])

Переводит регион в континент. В остальном, аналогично функции regionToCity.

Пример: regionToContinent(toUInt32(213)) = 10001 – преобразовали Москву (213) в Евразию (10001).

regionToTopContinent (#regiontotopcontinent)

Находит для региона верхний в иерархии континент.

Синтаксис:

regionToTopContinent(id[, geobase]);

Параметры:

* id – Идентификатор региона из геобазы. [UInt32](https://clickhouse.tech/docs/ru/sql-reference/functions/ym-dict-functions/).
* geobase – Ключ словаря. Смотрите [Множественные геобазы](https://clickhouse.tech/docs/ru/sql-reference/functions/ym-dict-functions/#multiple-geobases). [String](https://clickhouse.tech/docs/ru/sql-reference/functions/ym-dict-functions/). Опциональный параметр.

Возвращаемое значение:

* Идентификатор континента верхнего уровня (последний при подъеме по иерархии регионов).
* 0, если его нет.

Тип: UInt32.

regionToPopulation(id[, geobase])

Получает население для региона.

Население может быть прописано в файлах с геобазой. Смотрите в разделе «Встроенные словари».

Если для региона не прописано население, возвращается 0.

В геобазе, население может быть прописано для дочерних регионов, но не прописано для родительских.

regionIn(lhs, rhs[, geobase])

Проверяет принадлежность региона lhs региону rhs. Возвращает число типа UInt8, равное 1, если принадлежит и 0, если не принадлежит.

Отношение рефлексивное – любой регион принадлежит также самому себе.

regionHierarchy(id[, geobase])

Принимает число типа UInt32 – идентификатор региона из геобазы. Возвращает массив идентификаторов регионов, состоящий из переданного региона и всех родителей по цепочке.

Пример: regionHierarchy(toUInt32(213)) = [213,1,3,225,10001,10000].

regionToName(id[, lang])

Принимает число типа UInt32 – идентификатор региона из геобазы. Вторым аргументом может быть передана строка – название языка. Поддерживаются языки ru, en, ua, uk, by, kz, tr. Если второй аргумент отсутствует - используется язык ru. Если язык не поддерживается – кидается исключение. Возвращает строку - название региона на соответствующем языке. Если региона с указанным идентификатором не существует - возвращается пустая строка.

ua и uk обозначают одно и то же – украинский язык.

##### Функции для реализации оператора IN

###### in, notIn, globalIn, globalNotIn

Смотрите раздел [Операторы IN](https://clickhouse.tech/docs/ru/sql-reference/operators/in/#select-in-operators).

##### Функция ArrayJoin

Это совсем необычная функция.

Обычные функции не изменяют множество строк, а лишь изменяют значения в каждой строке (map).

Агрегатные функции выполняют свёртку множества строк (fold, reduce).

Функция arrayJoin выполняет размножение каждой строки в множество строк (unfold).

Функция принимает в качестве аргумента массив, и размножает исходную строку в несколько строк – по числу элементов массива.

Все значения в столбцах просто копируются, кроме значения в столбце с применением этой функции – он заменяется на соответствующее значение массива.

В запросе может быть использовано несколько функций arrayJoin. В этом случае, соответствующее преобразование делается несколько раз.

Обратите внимание на синтаксис ARRAY JOIN в запросе SELECT, который предоставляет более широкие возможности.

Пример:

**SELECT** arrayJoin([1, 2, 3] **AS** src) **AS** dst, 'Hello', src

┌─dst─┬─\'Hello\'─┬─src─────┐

│ 1 │ Hello │ [1,2,3] │

│ 2 │ Hello │ [1,2,3] │

│ 3 │ Hello │ [1,2,3] │

└─────┴───────────┴─────────┘

##### Функции для работы с географическими координатами

###### greatCircleDistance

Вычисляет расстояние между двумя точками на поверхности Земли по [формуле большого круга](https://en.wikipedia.org/wiki/Great-circle_distance).

Синтаксис:

greatCircleDistance(lon1Deg, lat1Deg, lon2Deg, lat2Deg)

Входные параметры:

* lon1Deg – долгота первой точки в градусах. Диапазон – [-180°, 180°].
* lat1Deg – широта первой точки в градусах. Диапазон – [-90°, 90°].
* lon2Deg – долгота второй точки в градусах. Диапазон – [-180°, 180°].
* lat2Deg – широта второй точки в градусах. Диапазон – [-90°, 90°].

Положительные значения соответствуют северной широте и восточной долготе, отрицательные – южной широте и западной долготе.

Возвращаемое значение:

* Расстояние между двумя точками на поверхности Земли в метрах.

Генерирует исключение, когда значения входных параметров выходят за границы диапазонов.

Пример:

**SELECT** greatCircleDistance(55.755831, 37.617673, -55.755831, -37.617673)

┌─greatCircleDistance(55.755831, 37.617673, -55.755831, -37.617673)─┐

│ 14132374.194975413 │

└───────────────────────────────────────────────────────────────────┘

###### greatCircleAngle

Вычисляет угловое расстояние на сфере по [формуле большого круга](https://en.wikipedia.org/wiki/Great-circle_distance).

Синтаксис:

greatCircleAngle(lon1Deg, lat1Deg, lon2Deg, lat2Deg)

Входные параметры:

* lon1Deg – долгота первой точки в градусах.
* lat1Deg – широта первой точки в градусах.
* lon2Deg – долгота второй точки в градусах.
* lat2Deg – широта второй точки в градусах.

Возвращаемое значение:

* Длина дуги большого круга между двумя точками в градусах.

Пример:

**SELECT** greatCircleAngle(0, 0, 45, 0) **AS** arc

┌─arc─┐

│ 45 │

└─────┘

###### pointInEllipses

Проверяет, принадлежит ли точка хотя бы одному из эллипсов.

Координаты – геометрические в декартовой системе координат.

Синтаксис:

pointInEllipses(x, y, x₀, y₀, a₀, b₀,...,xₙ, yₙ, aₙ, bₙ)

Входные параметры:

* x, y – координаты точки на плоскости.
* xᵢ, yᵢ – координаты центра i-го эллипса.
* aᵢ, bᵢ – полуоси i-го эллипса (в единицах измерения координат x,y).

Входных параметров должно быть 2+4⋅n, где n – количество эллипсов.

Возвращаемые значения:

1, если точка внутри хотя бы одного из эллипсов, 0, если нет.

Пример:

**SELECT** pointInEllipses(10., 10., 10., 9.1, 1., 0.9999)

┌─pointInEllipses(10., 10., 10., 9.1, 1., 0.9999)─┐

│ 1 │

└─────────────────────────────────────────────────┘

###### pointInPolygon

Проверяет, принадлежит ли точка многоугольнику на плоскости.

Синтаксис:

pointInPolygon((x, y), [(a, b), (**c**, d) ...], ...)

Входные значения:

* (x, y) – координаты точки на плоскости. Тип данных – [Tuple](https://clickhouse.tech/docs/ru/sql-reference/data-types/tuple/) – кортеж из двух чисел.
* [(a, b), (c, d) ...] – вершины многоугольника. Тип данных – [Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/). Каждая вершина представлена парой координат (a, b). Вершины следует указывать в порядке обхода по или против часовой стрелки. Минимальное количество вершин – 3. Многоугольник должен быть константным.
* функция поддерживает также многоугольники с дырками (вырезанными кусками). Для этого случая, добавьте многоугольники, описывающие вырезанные куски, дополнительными аргументами функции. Функция не поддерживает не односвязные многоугольники.

Возвращаемые значения:

* 1, если точка внутри многоугольника, 0, если нет.

Если точка находится на границе многоугольника, функция может возвращать как 0, так и 1.

Пример:

**SELECT** pointInPolygon((3., 3.), [(6, 0), (8, 4), (5, 8), (0, 2)]) **AS** res

┌─res─┐

│ 1 │

└─────┘

##### Функции для работы с индексами H3

[H3](https://eng.uber.com/h3/) – это система геокодирования, которая делит поверхность Земли на равные шестигранные ячейки. Система поддерживает иерархию (вложенность) ячеек, т.е. каждый "родительский" шестигранник может быть поделен на семь одинаковых вложенных "дочерних" шестигранников, и так далее.

Уровень вложенности назвается разрешением и может принимать значение от 0 до 15, где 0 соответствует базовым ячейкам самого верхнего уровня (наиболее крупным).

Для каждой точки, имеющей широту и долготу, можно получить 64-битный индекс H3, соответствующий номеру шестигранной ячейки, где эта точка находится.

Индексы H3 используются, в основном, для геопозиционирования и расчета расстояний.

###### h3IsValid

Проверяет корректность [H3](https://clickhouse.tech/docs/ru/sql-reference/functions/geo/h3/#h3index)-индекса.

Синтаксис:

h3IsValid(h3index)

Параметр:

* h3index – идентификатор шестигранника. Тип данных: [UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Возвращаемые значения:

* 1 – число является H3-индексом.
* 0 – число не является H3-индексом.

Тип: [UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Пример:

Запрос:

**SELECT** h3IsValid(630814730351855103) **as** h3IsValid

Результат:

┌─h3IsValid─┐

│ 1 │

└───────────┘

###### h3GetResolution

Извлекает разрешение [H3](https://clickhouse.tech/docs/ru/sql-reference/functions/geo/h3/#h3index)-индекса.

Синтаксис:

h3GetResolution(h3index)

Параметр:

* h3index – идентификатор шестигранника. Тип данных: [UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Возвращаемые значения:

* Разрешение сетки. Диапазон значений: [0, 15].
* Для несуществующего идентификатора может быть возвращено произвольное значение. Используйте [h3IsValid](https://clickhouse.tech/docs/ru/sql-reference/functions/geo/h3/#h3isvalid) для проверки идентификаторов.

Тип: [UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Пример:

Запрос:

**SELECT** h3GetResolution(639821929606596015) **as** resolution

Результат:

┌─resolution─┐

│ 14 │

└────────────┘

###### h3EdgeAngle

Рассчитывает средний размер стороны шестигранника [H3](https://clickhouse.tech/docs/ru/sql-reference/functions/geo/h3/#h3index) в градусах.

Синтаксис:

h3EdgeAngle(resolution)

Параметр:

* resolution – требуемое разрешение индекса. Тип данных: [UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/). Диапазон возможных значений: [0, 15].

Возвращаемое значение:

* Средняя длина стороны шестигранника [H3](https://clickhouse.tech/docs/ru/sql-reference/functions/geo/h3/#h3index) в градусах. Тип данных: [Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/).

Пример:

Запрос:

**SELECT** h3EdgeAngle(10) **as** edgeAngle

Результат:

┌───────h3EdgeAngle(10)─┐

│ 0.0005927224846720883 │

└───────────────────────┘

###### h3EdgeLengthM

Рассчитывает средний размер стороны шестигранника [H3](https://clickhouse.tech/docs/ru/sql-reference/functions/geo/h3/#h3index) в метрах.

Синтаксис:

h3EdgeLengthM(resolution)

Параметр:

* resolution – требуемое разрешение индекса. Тип данных – [UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/). Диапазон возможных значений – [0, 15].

Возвращаемое значение:

* Средняя длина стороны шестигранника H3 в метрах, тип – [Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/).

Пример:

Запрос:

**SELECT** h3EdgeLengthM(15) **as** edgeLengthM

Результат:

┌─edgeLengthM─┐

│ 0.509713273 │

└─────────────┘

###### geoToH3

Возвращает [H3](https://clickhouse.tech/docs/ru/sql-reference/functions/geo/h3/#h3index) индекс точки (lon, lat) с заданным разрешением.

Синтаксис:

geoToH3(lon, lat, resolution)

Параметры:

* lon – географическая долгота. Тип данных – [Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/).
* lat – географическая широта. Тип данных – [Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/).
* resolution – требуемое разрешение индекса. Тип данных – [UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/). Диапазон возможных значений – [0, 15].

Возвращаемые значения:

* Порядковый номер шестигранника.
* 0 в случае ошибки.

Тип данных: [UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Пример:

Запрос:

**SELECT** geoToH3(37.79506683, 55.71290588, 15) **as** h3Index

Ответ:

┌────────────h3Index─┐

│ 644325524701193974 │

└────────────────────┘

###### h3kRing

Возвращает [H3](https://clickhouse.tech/docs/ru/sql-reference/functions/geo/h3/#h3index)-индексы шестигранников в радиусе k от данного в произвольном порядке.

Синтаксис:

h3kRing(h3index, k)

Параметры:

* h3index – идентификатор шестигранника. Тип данных: [UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).
* k – радиус. Тип данных: [целое число](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)

Возвращаемые значения:

* Массив из H3-индексов.

Тип данных: [Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)).

Пример:

Запрос:

**SELECT** arrayJoin(h3kRing(644325529233966508, 1)) **AS** h3index

Результат:

┌────────────h3index─┐

│ 644325529233966508 │

│ 644325529233966497 │

│ 644325529233966510 │

│ 644325529233966504 │

│ 644325529233966509 │

│ 644325529233966355 │

│ 644325529233966354 │

└────────────────────┘

###### h3GetBaseCell

Определяет номер базовой (верхнеуровневой) шестиугольной [H3](https://clickhouse.tech/docs/ru/sql-reference/functions/geo/h3/#h3index)-ячейки для указанной ячейки.

Синтаксис:

h3GetBaseCell(**index**)

Параметр:

* index – индекс шестиугольной ячейки. Тип: [UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Возвращаемое значение:

* Индекс базовой шестиугольной ячейки.

Тип: [UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Пример:

Запрос:

**SELECT** h3GetBaseCell(612916788725809151) **as** basecell;

Результат:

┌─basecell─┐

│ 12 │

└──────────┘

###### h3HexAreaM2

Определяет среднюю площадь шестиугольной [H3](https://clickhouse.tech/docs/ru/sql-reference/functions/geo/h3/#h3index)-ячейки заданного разрешения в квадратных метрах.

Синтаксис:

h3HexAreaM2(resolution)

Параметр:

* resolution – разрешение. Диапазон: [0, 15]. Тип: [UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Возвращаемое значение:

* Площадь в квадратных метрах. Тип: [Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/).

Пример:

Запрос:

**SELECT** h3HexAreaM2(13) **as** area;

Результат:

┌─area─┐

│ 43.9 │

└──────┘

###### h3IndexesAreNeighbors

Определяет, являются ли [H3](https://clickhouse.tech/docs/ru/sql-reference/functions/geo/h3/#h3index)-ячейки соседями.

Синтаксис:

h3IndexesAreNeighbors(index1, index2)

Параметры:

* index1 – индекс шестиугольной ячейки. Тип: [UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).
* index2 – индекс шестиугольной ячейки. Тип: [UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Возвращаемое значение:

* 1 – ячейки являются соседями.
* 0 – ячейки не являются соседями.

Тип: [UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Пример:

Запрос:

**SELECT** h3IndexesAreNeighbors(617420388351344639, 617420388352655359) **AS** n;

Результат:

┌─n─┐

│ 1 │

└───┘

###### h3ToChildren

Формирует массив дочерних (вложенных) [H3](https://clickhouse.tech/docs/ru/sql-reference/functions/geo/h3/#h3index)-ячеек для указанной ячейки.

Синтаксис:

h3ToChildren(**index**, resolution)

Параметры:

* index – индекс шестиугольной ячейки. Тип: [UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).
* resolution – разрешение. Диапазон: [0, 15]. Тип: [UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Возвращаемое значение:

* Массив дочерних H3-ячеек.

Тип: [Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)([UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/)).

Пример:

Запрос:

**SELECT** h3ToChildren(599405990164561919, 6) **AS** children;

Результат:

┌─children───────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────┐

│ [603909588852408319,603909588986626047,603909589120843775,603909589255061503,603909589389279231,603909589523496959,603909589657714687] │

└────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────┘

###### h3ToParent

Определяет родительскую (более крупную) [H3](https://clickhouse.tech/docs/ru/sql-reference/functions/geo/h3/#h3index)-ячейку, содержащую указанную ячейку.

Синтаксис:

h3ToParent(**index**, resolution)

Параметры:

* index – индекс шестиугольной ячейки. Тип: [UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).
* resolution – разрешение. Диапазон: [0, 15]. Тип: [UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Возвращаемое значение:

* Индекс родительской H3-ячейки.

Тип: [UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Пример:

Запрос:

**SELECT** h3ToParent(599405990164561919, 3) **as** parent;

Результат:

┌─────────────parent─┐

│ 590398848891879423 │

└────────────────────┘

###### h3ToString

Преобразует [H3](https://clickhouse.tech/docs/ru/sql-reference/functions/geo/h3/#h3index)-индекс из числового представления H3Index в строковое.

Синтаксис:

h3ToString(**index**)

Параметр:

* index – индекс шестиугольной ячейки. Тип: [UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Возвращаемое значение:

* Строковое представление H3-индекса.

Тип: [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).

Пример:

Запрос:

**SELECT** h3ToString(617420388352917503) **as** h3\_string;

Результат:

┌─h3\_string───────┐

│ 89184926cdbffff │

└─────────────────┘

###### stringToH3

Преобразует [H3](https://clickhouse.tech/docs/ru/sql-reference/functions/geo/h3/#h3index)-индекс из строкового представления в числовое представление H3Index.

Синтаксис:

stringToH3(index\_str)

Параметр:

* index\_str – строковое представление H3-индекса. Тип: [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).

Возвращаемое значение:

* Числовое представление индекса шестиугольной ячейки.
* 0, если при преобразовании возникла ошибка.

Тип: [UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Пример:

Запрос:

**SELECT** stringToH3('89184926cc3ffff') **as** **index**;

Результат:

┌──────────────index─┐

│ 617420388351344639 │

└────────────────────┘

###### h3GetResolution

Определяет разрешение [H3](https://clickhouse.tech/docs/ru/sql-reference/functions/geo/h3/#h3index)-ячейки.

Синтаксис:

h3GetResolution(**index**)

Параметр:

* index – индекс шестиугольной ячейки. Тип: [UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Возвращаемое значение:

* Разрешение ячейки. Диапазон: [0, 15].

Тип: [UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Пример:

Запрос:

**SELECT** h3GetResolution(617420388352917503) **as** res;

Результат:

┌─res─┐

│ 9 │

└─────┘

##### Функции для работы с системой Geohash

[Geohash](https://en.wikipedia.org/wiki/Geohash) – это система геокодирования, которая делит поверхность Земли на участки в виде "решетки", и каждую ячейку решетки кодирует в виде строки из букв и цифр. Система поддерживает иерархию (вложенность) ячеек, поэтому чем точнее определена геопозиция, тем длиннее строка с кодом соответствующей ячейки.

Для ручного преобразования географических координат в строку geohash можно использовать сайт [geohash.org](http://geohash.org/).

###### geohashEncode

Кодирует широту и долготу в строку [geohash](https://clickhouse.tech/docs/ru/sql-reference/functions/geo/geohash/#geohash).

Синтаксис:

geohashEncode(longitude, latitude, [**precision**])

Входные значения:

* longitude – долгота. Диапазон – [-180°, 180°].
* latitude – широта. Диапазон – [-90°, 90°].
* precision – длина результирующей строки, по умолчанию 12. Опционально. Целое число в диапазоне [1, 12]. Любое значение меньше, чем 1 или больше 12 автоматически преобразуются в 12.

Возвращаемые значения:

* Строка с координатой, закодированной модифицированной версией алфавита base32.

Пример:

**SELECT** geohashEncode(-5.60302734375, 42.593994140625, 0) **AS** res

┌─res──────────┐

│ ezs42d000000 │

└──────────────┘

###### geohashDecode

Декодирует любую строку, закодированную в [geohash](https://clickhouse.tech/docs/ru/sql-reference/functions/geo/geohash/#geohash), на долготу и широту.

Синтаксис:

geohashDecode(geohash\_string)

Входные значения:

* geohash\_string – строка, содержащая geohash.

Возвращаемые значения:

* (longitude, latitude) – широта и долгота. Кортеж из двух значений типа Float64.

Пример:

**SELECT** geohashDecode('ezs42') **AS** res

┌─res─────────────────────────────┐

│ (-5.60302734375,42.60498046875) │

└─────────────────────────────────┘

###### geohashesInBox

Формирует массив участков, которые находятся внутри или пересекают границу заданного участка на поверхности. Каждый участок описывается строкой [geohash](https://clickhouse.tech/docs/ru/sql-reference/functions/geo/geohash/#geohash) заданной точности.

Синтаксис:

geohashesInBox(longitude\_min, latitude\_min, longitude\_max, latitude\_max, **precision**)

Параметры

* longitude\_min – минимальная долгота. Диапазон возможных значений: [-180°, 180°]. Тип данных: [Float](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/)).
* latitude\_min – минимальная широта. Диапазон возможных значений: [-90°, 90°]. Тип данных: [Float](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/).
* longitude\_max – максимальная долгота. Диапазон возможных значений: [-180°, 180°]. Тип данных: [Float](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/).
* latitude\_max – максимальная широта. Диапазон возможных значений: [-90°, 90°]. Тип данных: [Float](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/).
* precision – точность geohash. Диапазон возможных значений: [1, 12]. Тип данных: [UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

***Замечание****: Все передаваемые координаты должны быть одного и того же типа: либо Float32, либо Float64.*

Возвращаемые значения:

* Массив строк, описывающих участки, покрывающие заданный участок. Длина каждой строки соответствует точности geohash. Порядок строк – произвольный.
* [] - Если переданные минимальные значения широты и долготы больше соответствующих максимальных значений, функция возвращает пустой массив.

Тип данных: [Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)).

Замечание

Если возвращаемый массив содержит свыше 10 000 000 элементов, функция сгенерирует исключение.

Пример:

Запрос:

**SELECT** geohashesInBox(24.48, 40.56, 24.785, 40.81, 4) **AS** thasos

Результат:

┌─thasos──────────────────────────────────────┐

│ ['sx1q','sx1r','sx32','sx1w','sx1x','sx38'] │

└─────────────────────────────────────────────┘

##### Функции для работы с Nullable-аргументами

###### isNull

Проверяет является ли аргумент [NULL](https://clickhouse.tech/docs/ru/sql-reference/syntax/#null-literal).

Синтаксис:

**isNull**(x)

Синоним: ISNULL.

Параметры:

* x – значение с не составным типом данных.

Возвращаемое значение:

* 1, если x – NULL.
* 0, если x – не NULL.

Пример:

Входная таблица:

┌─x─┬────y─┐

│ 1 │ ᴺᵁᴸᴸ │

│ 2 │ 3 │

└───┴──────┘

Запрос:

**SELECT** x **FROM** t\_null **WHERE** **isNull**(y)

┌─x─┐

│ 1 │

└───┘

###### isNotNull

Проверяет не является ли аргумент [NULL](https://clickhouse.tech/docs/ru/sql-reference/syntax/#null-literal).

Синтаксис:

isNotNull(x)

Параметры:

* x – значение с не составным типом данных.

Возвращаемое значение:

* 0, если x – NULL.
* 1, если x – не NULL.

Пример:

Входная таблица:

┌─x─┬────y─┐

│ 1 │ ᴺᵁᴸᴸ │

│ 2 │ 3 │

└───┴──────┘

Запрос:

**SELECT** x **FROM** t\_null **WHERE** isNotNull(y)

┌─x─┐

│ 2 │

└───┘

###### coalesce

Последовательно слева-направо проверяет являются ли переданные аргументы NULL и возвращает первый не NULL.

Синтаксис:

coalesce(x,...)

Параметры

* Произвольное количество параметров не составного типа. Все параметры должны быть совместимы по типу данных.

Возвращаемые значения:

* Первый не NULL аргумент.
* NULL, если все аргументы – NULL.

Пример:

Рассмотрим адресную книгу, в которой может быть указано несколько способов связи с клиентом.

┌─name─────┬─mail─┬─phone─────┬──icq─┐

│ client 1 │ ᴺᵁᴸᴸ │ 123-45-67 │ 123 │

│ client 2 │ ᴺᵁᴸᴸ │ ᴺᵁᴸᴸ │ ᴺᵁᴸᴸ │

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

Поля mail и phone имеют тип String, а поле icq – UInt32, его необходимо будет преобразовать в String.

Получим из адресной книги первый доступный способ связаться с клиентом:

**SELECT** coalesce(mail, phone, **CAST**(icq,'Nullable(String)')) **FROM** aBook

┌─name─────┬─coalesce(mail, phone, CAST(icq, 'Nullable(String)'))─┐

│ client 1 │ 123-45-67 │

│ client 2 │ ᴺᵁᴸᴸ │

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

###### ifNull

Возвращает альтернативное значение, если основной аргумент – NULL.

Синтаксис:

ifNull(x,alt)

Параметры:

* x – значение для проверки на NULL,
* alt – значение, которое функция вернёт, если x – NULL.

Возвращаемые значения:

* Значение x, если x – не NULL.
* Значение alt, если x – NULL.

Пример:

**SELECT** ifNull('a', 'b')

┌─ifNull('a', 'b')─┐

│ a │

└──────────────────┘

**SELECT** ifNull(**NULL**, 'b')

┌─ifNull(NULL, 'b')─┐

│ b │

└───────────────────┘

###### nullIf

Возвращает NULL, если аргументы равны.

Синтаксис:

**nullIf**(x, y)

Параметры:

x, y – значения для сравнивания. Они должны быть совместимых типов, иначе RT.WideStore сгенерирует исключение.

Возвращаемые значения:

* NULL, если аргументы равны.
* Значение x, если аргументы не равны.

Пример:

**SELECT** **nullIf**(1, 1)

┌─nullIf(1, 1)─┐

│ ᴺᵁᴸᴸ │

└──────────────┘

**SELECT** **nullIf**(1, 2)

┌─nullIf(1, 2)─┐

│ 1 │

└──────────────┘

###### assumeNotNull

Приводит значение типа [Nullable](https://clickhouse.tech/docs/ru/sql-reference/functions/functions-for-nulls/) к не Nullable, если значение не NULL.

Синтаксис:

assumeNotNull(x)

Параметры:

* x – исходное значение.

Возвращаемые значения:

* Исходное значение с не Nullable типом, если оно – не NULL.
* Значение по умолчанию для не Nullable типа, если исходное значение – NULL.

Пример:

Рассмотрим таблицу t\_null.

**SHOW** **CREATE** **TABLE** t\_null

┌─statement─────────────────────────────────────────────────────────────────┐

│ CREATE TABLE default.t\_null ( x Int8, y Nullable(Int8)) ENGINE = TinyLog │

└───────────────────────────────────────────────────────────────────────────┘

┌─x─┬────y─┐

│ 1 │ ᴺᵁᴸᴸ │

│ 2 │ 3 │

└───┴──────┘

Применим функцию assumeNotNull к столбцу y.

**SELECT** assumeNotNull(y) **FROM** t\_null

┌─assumeNotNull(y)─┐

│ 0 │

│ 3 │

└──────────────────┘

**SELECT** toTypeName(assumeNotNull(y)) **FROM** t\_null

┌─toTypeName(assumeNotNull(y))─┐

│ Int8 │

│ Int8 │

└──────────────────────────────┘

###### toNullable

Преобразует тип аргумента к Nullable.

Синтаксис:

toNullable(x)

Параметры:

* x – значение произвольного не составного типа.

Возвращаемое значение значение:

* Входное значение с типом не Nullable.

Пример:

**SELECT** toTypeName(10)

┌─toTypeName(10)─┐

│ UInt8 │

└────────────────┘

**SELECT** toTypeName(toNullable(10))

┌─toTypeName(toNullable(10))─┐

│ Nullable(UInt8) │

└────────────────────────────┘

##### Функции машинного обучения

###### evalMLMethod (prediction)

Предсказание с использованием подобранных регрессионных моделей.

Stochastic Linear Regression

Агрегатная функция [stochasticLinearRegression](https://clickhouse.tech/docs/ru/sql-reference/functions/machine-learning-functions/#agg_functions-stochasticlinearregression) реализует стохастический градиентный спуск, использую линейную модель и функцию потерь MSE.

Stochastic Logistic Regression

Агрегатная функция [stochasticLogisticRegression](https://clickhouse.tech/docs/ru/sql-reference/functions/machine-learning-functions/#agg_functions-stochasticlogisticregression) реализует стохастический градиентный спуск для задачи бинарной классификации.

###### bayesAB

Сравнивает тестовые группы (варианты) и для каждой группы рассчитывает вероятность того, что эта группа окажется лучшей. Первая из перечисленных групп считается контрольной.

Синтаксис:

bayesAB(distribution\_name, higher\_is\_better, variant\_names, x, y)

Параметры

* distribution\_name – вероятностное распределение. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/). Возможные значения:
  + beta для [Бета-распределения](https://ru.wikipedia.org/wiki/%D0%91%D0%B5%D1%82%D0%B0-%D1%80%D0%B0%D1%81%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5).
  + gamma для [Гамма-распределения](https://ru.wikipedia.org/wiki/%D0%93%D0%B0%D0%BC%D0%BC%D0%B0-%D1%80%D0%B0%D1%81%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5).
* higher\_is\_better – способ определения предпочтений. [Boolean](https://clickhouse.tech/docs/ru/sql-reference/data-types/boolean/). Возможные значения:
  + 0 – чем меньше значение, тем лучше.
  + 1 – чем больше значение, тем лучше.
* variant\_names – массив, содержащий названия вариантов. [Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)([String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/)).
* x – массив, содержащий число проведенных тестов (испытаний) для каждого варианта. [Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)([Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/)).
* y – массив, содержащий число успешных тестов (испытаний) для каждого варианта. [Array](https://clickhouse.tech/docs/ru/sql-reference/data-types/array/)([Float64](https://clickhouse.tech/docs/ru/sql-reference/data-types/float/)).

***Замечание:*** *Все три массива должны иметь одинаковый размер. Все значения x и y должны быть неотрицательными числами (константами). Значение y не может превышать соответствующее значение x.*

Возвращаемые значения:

Для каждого варианта рассчитываются:

* beats\_control – вероятность, что данный вариант превосходит контрольный в долгосрочной перспективе.
* to\_be\_best – вероятность, что данный вариант является лучшим в долгосрочной перспективе.

Тип: JSON.

Пример:

Запрос:

**SELECT** bayesAB('beta', 1, ['Control', 'A', 'B'], [3000., 3000., 3000.], [100., 90., 110.]) FORMAT PrettySpace;

Результат:

{

"data":[

{

"variant\_name":"Control",

"x":3000,

"y":100,

"beats\_control":0,

"to\_be\_best":0.22619

},

{

"variant\_name":"A",

"x":3000,

"y":90,

"beats\_control":0.23469,

"to\_be\_best":0.04671

},

{

"variant\_name":"B",

"x":3000,

"y":110,

"beats\_control":0.7580899999999999,

"to\_be\_best":0.7271

}

]

}

##### Функции интроспекции

Функции из этого раздела могут использоваться для интроспекции [ELF](https://en.wikipedia.org/wiki/Executable_and_Linkable_Format) и [DWARF](https://en.wikipedia.org/wiki/DWARF) в целях профилирования запросов.

Предупреждение

Эти функции выполняются медленно и могут приводить к нежелательным последствиям в плане безопасности.

Для правильной работы функций интроспекции:

* Установите пакет clickhouse-common-static-dbg.
* Установите настройку [allow\_introspection\_functions](https://clickhouse.tech/docs/ru/operations/settings/settings/#settings-allow_introspection_functions) в 1.

Из соображений безопасности данные функции отключены по умолчанию.

RT.WideStore сохраняет отчеты профилировщика в [журнал трассировки](https://clickhouse.tech/docs/ru/operations/system-tables/trace_log/#system_tables-trace_log) в системной таблице. Убедитесь, что таблица и профилировщик настроены правильно.

###### addresssToLine

Преобразует адрес виртуальной памяти внутри процесса сервера RT.WideStore в имя файла и номер строки в исходном коде RT.WideStore.

Если вы используете официальные пакеты RT.WideStore, вам необходимо установить следующий пакеты: clickhouse-common-static-dbg.

Синтаксис:

addressToLine(address\_of\_binary\_instruction)

Параметры:

* address\_of\_binary\_instruction ([Тип UInt64](https://clickhouse.tech/docs/ru/sql-reference/functions/introspection/)) – адрес инструкции в запущенном процессе.

Возвращаемое значение:

* Имя файла исходного кода и номер строки в этом файле разделяются двоеточием.

Например, `/build/obj-x86\_64-linux-gnu/../src/Common/ThreadPool.cpp:199`, где `199` – номер строки.

* Имя бинарного файла, если функция не может найти отладочную информацию.
* Пустая строка, если адрес не является допустимым.

Тип: [String](https://clickhouse.tech/docs/ru/sql-reference/functions/introspection/).

Пример:

Включение функций самоанализа:

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

Выбор первой строки из списка trace\_log системная таблица:

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

Row 1:

──────

event\_date: 2019-11-19

event\_time: 2019-11-19 18:57:23

revision: 54429

timer\_type: Real

thread\_number: 48

query\_id: 421b6855-1858-45a5-8f37-f383409d6d72

trace: [140658411141617,94784174532828,94784076370703,94784076372094,94784076361020,94784175007680,140658411116251,140658403895439]

То trace поле содержит трассировку стека в момент выборки.

Получение имени файла исходного кода и номера строки для одного адреса:

**SELECT** addressToLine(94784076370703) \**G**

Row 1:

──────

addressToLine(94784076370703): /build/obj-x86\_64-linux-gnu/../src/Common/ThreadPool.cpp:199

Применение функции ко всему стектрейсу:

**SELECT**

arrayStringConcat(arrayMap(x -> addressToLine(x), trace), '\n') **AS** trace\_source\_code\_lines

**FROM** **system**.trace\_log

**LIMIT** 1

\**G**

Функция [arrayMap](https://clickhouse.tech/docs/ru/sql-reference/functions/array-functions/#array-map) позволяет обрабатывать каждый отдельный элемент массива trace с помощью функции addressToLine. Результат этой обработки вы видите в виде trace\_source\_code\_lines колонки выходных данных.

Row 1:

──────

trace\_source\_code\_lines: /lib/x86\_64-linux-gnu/libpthread-2.27.so

/usr/lib/debug/usr/bin/clickhouse

/build/obj-x86\_64-linux-gnu/../src/Common/ThreadPool.cpp:199

/build/obj-x86\_64-linux-gnu/../src/Common/ThreadPool.h:155

/usr/include/c++/9/bits/atomic\_base.h:551

/usr/lib/debug/usr/bin/clickhouse

/lib/x86\_64-linux-gnu/libpthread-2.27.so

/build/glibc-OTsEL5/glibc-2.27/misc/../sysdeps/unix/sysv/linux/x86\_64/clone.S:97

###### addressToSymbol

Преобразует адрес виртуальной памяти внутри серверного процесса ClickHouse в символ из объектных файлов RT.WideStore.

Синтаксис:

addressToSymbol(address\_of\_binary\_instruction)

Параметры

* address\_of\_binary\_instruction ([Тип uint64](https://clickhouse.tech/docs/ru/sql-reference/functions/introspection/)) – адрес инструкции в запущенном процессе.

Возвращаемое значение:

* Символ из объектных файлов RT.WideStore.
* Пустая строка, если адрес не является допустимым.

Тип: [String](https://clickhouse.tech/docs/ru/sql-reference/functions/introspection/).

Пример:

Включение функций самоанализа:

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

Выбор первой строки из списка trace\_log системная таблица:

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

Row 1:

──────

event\_date: 2019-11-20

event\_time: 2019-11-20 16:57:59

revision: 54429

timer\_type: Real

thread\_number: 48

query\_id: 724028bf-f550-45aa-910d-2af6212b94ac

trace: [94138803686098,94138815010911,94138815096522,94138815101224,94138815102091,94138814222988,94138806823642,94138814457211,94138806823642,94138814457211,94138806823642,94138806795179,94138806796144,94138753770094,94138753771646,94138753760572,94138852407232,140399185266395,140399178045583]

То trace поле содержит трассировку стека в момент выборки.

Получение символа для одного адреса:

**SELECT** addressToSymbol(94138803686098) \**G**

Row 1:

──────

addressToSymbol(94138803686098): \_ZNK2DB24IAggregateFunctionHelperINS\_20AggregateFunctionSumImmNS\_24AggregateFunctionSumDataImEEEEE19addBatchSinglePlaceEmPcPPKNS\_7IColumnEPNS\_5ArenaE

Применение функции ко всей трассировке стека:

**SELECT**

arrayStringConcat(arrayMap(x -> addressToSymbol(x), trace), '\n') **AS** trace\_symbols

**FROM** **system**.trace\_log

**LIMIT** 1

\**G**

То [arrayMap](https://clickhouse.tech/docs/ru/sql-reference/functions/array-functions/#array-map) функция позволяет обрабатывать каждый отдельный элемент системы. trace массив по типу addressToSymbols функция. Результат этой обработки вы видите в виде trace\_symbols колонка выходных данных.

Row 1:

──────

trace\_symbols: \_ZNK2DB24IAggregateFunctionHelperINS\_20AggregateFunctionSumImmNS\_24AggregateFunctionSumDataImEEEEE19addBatchSinglePlaceEmPcPPKNS\_7IColumnEPNS\_5ArenaE

\_ZNK2DB10Aggregator21executeWithoutKeyImplERPcmPNS0\_28AggregateFunctionInstructionEPNS\_5ArenaE

\_ZN2DB10Aggregator14executeOnBlockESt6vectorIN3COWINS\_7IColumnEE13immutable\_ptrIS3\_EESaIS6\_EEmRNS\_22AggregatedDataVariantsERS1\_IPKS3\_SaISC\_EERS1\_ISE\_SaISE\_EERb

\_ZN2DB10Aggregator14executeOnBlockERKNS\_5BlockERNS\_22AggregatedDataVariantsERSt6vectorIPKNS\_7IColumnESaIS9\_EERS6\_ISB\_SaISB\_EERb

\_ZN2DB10Aggregator7executeERKSt10shared\_ptrINS\_17IBlockInputStreamEERNS\_22AggregatedDataVariantsE

\_ZN2DB27AggregatingBlockInputStream8readImplEv

\_ZN2DB17IBlockInputStream4readEv

\_ZN2DB26ExpressionBlockInputStream8readImplEv

\_ZN2DB17IBlockInputStream4readEv

\_ZN2DB26ExpressionBlockInputStream8readImplEv

\_ZN2DB17IBlockInputStream4readEv

\_ZN2DB28AsynchronousBlockInputStream9calculateEv

\_ZNSt17\_Function\_handlerIFvvEZN2DB28AsynchronousBlockInputStream4nextEvEUlvE\_E9\_M\_invokeERKSt9\_Any\_data

\_ZN14ThreadPoolImplI20ThreadFromGlobalPoolE6workerESt14\_List\_iteratorIS0\_E

\_ZZN20ThreadFromGlobalPoolC4IZN14ThreadPoolImplIS\_E12scheduleImplIvEET\_St8functionIFvvEEiSt8optionalImEEUlvE1\_JEEEOS4\_DpOT0\_ENKUlvE\_clEv

\_ZN14ThreadPoolImplISt6threadE6workerESt14\_List\_iteratorIS0\_E

execute\_native\_thread\_routine

start\_thread

clone

###### demangle

Преобразует символ, который вы можете получить с помощью [addressstosymbol](https://clickhouse.tech/docs/ru/sql-reference/functions/introspection/#addresstosymbol) функция имя функции C++.

Синтаксис:

demangle(symbol)

Параметры:

* symbol ([Строка](https://clickhouse.tech/docs/ru/sql-reference/functions/introspection/)) – символ из объектного файла.

Возвращаемое значение:

* Имя функции C++.
* Пустая строка, если символ не является допустимым.

Тип: [Строка](https://clickhouse.tech/docs/ru/sql-reference/functions/introspection/).

Пример:

Включение функций самоанализа:

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

Выбор первой строки из списка trace\_log системная таблица:

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

Row 1:

──────

event\_date: 2019-11-20

event\_time: 2019-11-20 16:57:59

revision: 54429

timer\_type: Real

thread\_number: 48

query\_id: 724028bf-f550-45aa-910d-2af6212b94ac

trace: [94138803686098,94138815010911,94138815096522,94138815101224,94138815102091,94138814222988,94138806823642,94138814457211,94138806823642,94138814457211,94138806823642,94138806795179,94138806796144,94138753770094,94138753771646,94138753760572,94138852407232,140399185266395,140399178045583]

То trace поле содержит трассировку стека в момент выборки.

Получение имени функции для одного адреса:

**SELECT** demangle(addressToSymbol(94138803686098)) \**G**

Row 1:

──────

demangle(addressToSymbol(94138803686098)): DB::IAggregateFunctionHelper<DB::AggregateFunctionSum<unsigned long, unsigned long, DB::AggregateFunctionSumData<unsigned long> > >::addBatchSinglePlace(unsigned long, char\*, DB::IColumn const\*\*, DB::Arena\*) const

Применение функции ко всему стектрейсу:

**SELECT**

arrayStringConcat(arrayMap(x -> demangle(addressToSymbol(x)), trace), '\n') **AS** trace\_functions

**FROM** **system**.trace\_log

**LIMIT** 1

\**G**

Функция [arrayMap](https://clickhouse.tech/docs/ru/sql-reference/functions/array-functions/#array-map) позволяет обрабатывать каждый отдельный элемент массива trace с помощью функции demangle.

Row 1:

──────

trace\_functions: DB::IAggregateFunctionHelper<DB::AggregateFunctionSum<unsigned long, unsigned long, DB::AggregateFunctionSumData<unsigned long> > >::addBatchSinglePlace(unsigned long, char\*, DB::IColumn const\*\*, DB::Arena\*) const

DB::Aggregator::executeWithoutKeyImpl(char\*&, unsigned long, DB::Aggregator::AggregateFunctionInstruction\*, DB::Arena\*) const

DB::Aggregator::executeOnBlock(std::vector<COW<DB::IColumn>::immutable\_ptr<DB::IColumn>, std::allocator<COW<DB::IColumn>::immutable\_ptr<DB::IColumn> > >, unsigned long, DB::AggregatedDataVariants&, std::vector<DB::IColumn const\*, std::allocator<DB::IColumn const\*> >&, std::vector<std::vector<DB::IColumn const\*, std::allocator<DB::IColumn const\*> >, std::allocator<std::vector<DB::IColumn const\*, std::allocator<DB::IColumn const\*> > > >&, bool&)

DB::Aggregator::executeOnBlock(DB::Block const&, DB::AggregatedDataVariants&, std::vector<DB::IColumn const\*, std::allocator<DB::IColumn const\*> >&, std::vector<std::vector<DB::IColumn const\*, std::allocator<DB::IColumn const\*> >, std::allocator<std::vector<DB::IColumn const\*, std::allocator<DB::IColumn const\*> > > >&, bool&)

DB::Aggregator::execute(std::shared\_ptr<DB::IBlockInputStream> const&, DB::AggregatedDataVariants&)

DB::AggregatingBlockInputStream::readImpl()

DB::IBlockInputStream::read()

DB::ExpressionBlockInputStream::readImpl()

DB::IBlockInputStream::read()

DB::ExpressionBlockInputStream::readImpl()

DB::IBlockInputStream::read()

DB::AsynchronousBlockInputStream::calculate()

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

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

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

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

execute\_native\_thread\_routine

start\_thread

clone

###### tid

Возвращает id потока, в котором обрабатывается текущий [Block](https://clickhouse.tech/docs/ru/development/architecture/#block).

Синтаксис:

tid()

Возвращаемое значение:

* Id текущего потока. [Uint64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges).

Пример:

Запрос:

**SELECT** tid();

Результат:

┌─tid()─┐

│ 3878 │

└───────┘

###### logTrace

Выводит сообщение в лог сервера для каждого [Block](https://clickhouse.tech/docs/ru/development/architecture/#block).

Синтаксис:

logTrace('message')

Параметры

* message – сообщение, которое отправляется в серверный лог. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/#string).

Возвращаемое значение:

* Всегда возвращает 0.

Пример:

Запрос:

**SELECT** logTrace('logTrace message');

Результат:

┌─logTrace('logTrace message')─┐

│ 0 │

└──────────────────────────────┘

##### Прочие функции

###### hostName()

Возвращает строку - имя хоста, на котором эта функция была выполнена. При распределённой обработке запроса, это будет имя хоста удалённого сервера, если функция выполняется на удалённом сервере.

###### getMacro

Возвращает именованное значение из секции [macros](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/#macros) конфигурации сервера.

Синтаксис:

getMacro(name);

Параметры:

* name – имя, которое необходимо получить из секции macros. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/#string).

Возвращаемое значение:

* Значение по указанному имени.

Тип: [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).

Пример:

Пример секции macros в конфигурационном файле сервера:

<macros>

<test>Value</test>

</macros>

Запрос:

**SELECT** getMacro('test');

Результат:

┌─getMacro('test')─┐

│ Value │

└──────────────────┘

Альтернативный способ получения значения:

**SELECT** \* **FROM** **system**.macros

**WHERE** macro = 'test'

┌─macro─┬─substitution─┐

│ test │ Value │

└───────┴──────────────┘

###### FQDN

Возвращает полное имя домена.

Синтаксис:

fqdn();

Эта функция регистронезависимая.

Возвращаемое значение:

* Полное имя домена.

Тип: String.

Пример:

Запрос:

**SELECT** FQDN();

Ответ:

┌─FQDN()──────────────────────────┐

│ clickhouse.ru-central1.internal │

└─────────────────────────────────┘

###### basename

Извлекает конечную часть строки после последнего слэша или бэкслэша. Функция часто используется для извлечения имени файла из пути.

Синтаксис:

basename( expr )

Параметры:

* expr – выражение, возвращающее значение типа [String](https://clickhouse.tech/docs/ru/sql-reference/functions/other-functions/). В результирующем значении все бэкслэши должны быть экранированы.

Возвращаемое значение:

Строка, содержащая:

* Конечную часть строки после последнего слэша или бэкслэша.

Если входная строка содержит путь, заканчивающийся слэшем или бэкслэшем, например, `/` или `с:\`, функция возвращает пустую строку.

* Исходная строка, если нет слэша или бэкслэша.

Пример:

**SELECT** 'some/long/path/to/file' **AS** a, basename(a)

┌─a──────────────────────┬─basename('some\\long\\path\\to\\file')─┐

│ some\long\path\to\file │ file │

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

**SELECT** 'some\\long\\path\\to\\file' **AS** a, basename(a)

┌─a──────────────────────┬─basename('some\\long\\path\\to\\file')─┐

│ some\long\path\to\file │ file │

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

**SELECT** 'some-file-name' **AS** a, basename(a)

┌─a──────────────┬─basename('some-file-name')─┐

│ some-file-name │ some-file-name │

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

###### visibleWidth(x)

Вычисляет приблизительную ширину при выводе значения в текстовом (tab-separated) виде на консоль.

Функция используется системой для реализации Pretty форматов.

NULL представляется как строка, соответствующая отображению NULL в форматах Pretty.

**SELECT** visibleWidth(**NULL**)

┌─visibleWidth(NULL)─┐

│ 4 │

└────────────────────┘

###### toTypeName(x)

Возвращает строку, содержащую имя типа переданного аргумента.

Если на вход функции передать NULL, то она вернёт тип Nullable(Nothing), что соответствует внутреннему представлению NULL в RT.WideStore.

###### blockSize()

Получить размер блока.

В RT.WideStore выполнение запроса всегда идёт по блокам (наборам кусочков столбцов). Функция позволяет получить размер блока, для которого её вызвали.

###### byteSize

Возвращает оценку в байтах размера аргументов в памяти в несжатом виде.

Синтаксис:

byteSize(argument [, ...])

Параметры:

* argument – значение.

Возвращаемое значение:

* Оценка размера аргументов в памяти в байтах.

Тип: [UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Примеры:

Для аргументов типа [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/) функция возвращает длину строки + 9 (нуль-терминатор + длина).

Запрос:

**SELECT** byteSize('string');

Результат:

┌─byteSize('string')─┐

│ 15 │

└────────────────────┘

Запрос:

**CREATE** **TABLE** test

(

`**key**` Int32,

`u8` UInt8,

`u16` UInt16,

`u32` UInt32,

`u64` UInt64,

`i8` Int8,

`i16` Int16,

`i32` Int32,

`i64` Int64,

`f32` Float32,

`f64` Float64

)

ENGINE = MergeTree

**ORDER** **BY** **key**;

**INSERT** **INTO** test **VALUES**(1, 8, 16, 32, 64, -8, -16, -32, -64, 32.32, 64.64);

**SELECT** **key**, byteSize(u8) **AS** `byteSize(UInt8)`, byteSize(u16) **AS** `byteSize(UInt16)`, byteSize(u32) **AS** `byteSize(UInt32)`, byteSize(u64) **AS** `byteSize(UInt64)`, byteSize(i8) **AS** `byteSize(Int8)`, byteSize(i16) **AS** `byteSize(Int16)`, byteSize(i32) **AS** `byteSize(Int32)`, byteSize(i64) **AS** `byteSize(Int64)`, byteSize(f32) **AS** `byteSize(Float32)`, byteSize(f64) **AS** `byteSize(Float64)` **FROM** test **ORDER** **BY** **key** **ASC** FORMAT Vertical;

Результат:

Row 1:

──────

key: 1

byteSize(UInt8): 1

byteSize(UInt16): 2

byteSize(UInt32): 4

byteSize(UInt64): 8

byteSize(Int8): 1

byteSize(Int16): 2

byteSize(Int32): 4

byteSize(Int64): 8

byteSize(Float32): 4

byteSize(Float64): 8

Если функция принимает несколько аргументов, то она возвращает их совокупный размер в байтах.

Запрос:

**SELECT** byteSize(**NULL**, 1, 0.3, '');

Результат:

┌─byteSize(NULL, 1, 0.3, '')─┐

│ 19 │

└────────────────────────────┘

###### materialize(x)

Превращает константу в полноценный столбец, содержащий только одно значение.

В RT.WideStore полноценные столбцы и константы представлены в памяти по-разному. Функции по-разному работают для аргументов-констант и обычных аргументов (выполняется разный код), хотя результат почти всегда должен быть одинаковым. Эта функция предназначена для отладки такого поведения.

###### ignore(…)

Принимает любые аргументы, в т.ч. NULL, всегда возвращает 0.

При этом, аргумент всё равно вычисляется. Это может использоваться для бенчмарков.

###### sleep(seconds)

Спит seconds секунд на каждый блок данных. Можно указать как целое число, так и число с плавающей запятой.

###### currentDatabase()

Возвращает имя текущей базы данных.

Эта функция может использоваться в параметрах движка таблицы в запросе CREATE TABLE там, где нужно указать базу данных.

###### currentUser()

Возвращает логин текущего пользователя. При распределенном запросе, возвращается имя пользователя, инициировавшего запрос.

Синтаксис:

**SELECT** currentUser();

Алиас: user(), USER().

Возвращаемые значения:

* Логин текущего пользователя.
* При распределенном запросе – логин пользователя, инициировавшего запрос.

Тип: String.

Пример:

Запрос:

**SELECT** currentUser();

Ответ:

┌─currentUser()─┐

│ default │

└───────────────┘

###### isConstant

Проверяет, является ли аргумент константным выражением.

Константное выражение – это выражение, результат которого известен на момент анализа запроса (до его выполнения). Например, выражения над [литералами](https://clickhouse.tech/docs/ru/sql-reference/syntax/#literals) являются константными.

Используется в целях разработки, отладки или демонстрирования.

Синтаксис:

isConstant(x)

Параметры:

* x – выражение для проверки.

Возвращаемые значения:

* 1 – выражение x является константным.
* 0 – выражение x не является константным.

Тип: [UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Примеры:

Запрос:

**SELECT** isConstant(x + 1) **FROM** (**SELECT** 43 **AS** x)

Результат:

┌─isConstant(plus(x, 1))─┐

│ 1 │

└────────────────────────┘

Запрос:

**WITH** 3.14 **AS** pi **SELECT** isConstant(cos(pi))

Результат:

┌─isConstant(cos(pi))─┐

│ 1 │

└─────────────────────┘

Запрос:

**SELECT** isConstant(number) **FROM** numbers(1)

Результат:

┌─isConstant(number)─┐

│ 0 │

└────────────────────┘

###### isFinite(x)

Принимает Float32 или Float64 и возвращает UInt8, равный 1, если аргумент не бесконечный и не NaN, иначе 0.

###### ifNotFinite

Проверяет, является ли значение дробного числа с плавающей точкой конечным.

Синтаксис:

ifNotFinite(x,y)

Параметры:

* x – значение, которое нужно проверить на бесконечность. Тип: [Float\*](https://clickhouse.tech/docs/ru/sql-reference/functions/other-functions/).
* y – запасное значение. Тип: [Float\*](https://clickhouse.tech/docs/ru/sql-reference/functions/other-functions/).

Возвращаемые значения:

* x, если x принимает конечное значение.
* y, если x принимает не конечное значение.

Пример:

Запрос:

SELECT 1/0 as infimum, ifNotFinite(infimum,42)

Результат:

┌─infimum─┬─ifNotFinite(divide(1, 0), 42)─┐

│ inf │ 42 │

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

Аналогичный результат можно получить с помощью [тернарного оператора](https://clickhouse.tech/docs/ru/sql-reference/functions/conditional-functions/#ternary-operator) isFinite(x) ? x : y.

###### isInfinite(x)

Принимает Float32 или Float64 и возвращает UInt8, равный 1, если аргумент бесконечный, иначе 0. Отметим, что в случае NaN возвращается 0.

###### isNaN(x)

Принимает Float32 или Float64 и возвращает UInt8, равный 1, если аргумент является NaN, иначе 0.

###### hasColumnInTable([‘hostname’[, ‘username’[, ‘password’]],] ‘database’, ‘table’, ‘column’)

Принимает константные строки - имя базы данных, имя таблицы и название столбца. Возвращает константное выражение типа UInt8, равное 1,

если есть столбец, иначе 0. Если задан параметр hostname, проверка будет выполнена на удалённом сервере.

Функция кидает исключение, если таблица не существует.

Для элементов вложенной структуры данных функция проверяет существование столбца. Для самой же вложенной структуры данных функция возвращает 0.

###### bar

Позволяет построить unicode-art диаграмму.

bar(x, min, max, width) рисует полосу ширины пропорциональной (x - min) и равной width символов при x = max.

Параметры:

* x – величина для отображения.
* min, max – целочисленные константы, значение должно помещаться в Int64.
* width – константа, положительное число, может быть дробным.

Полоса рисуется с точностью до одной восьмой символа.

Пример:

**SELECT**

toHour(EventTime) **AS** h,

**count**() **AS** **c**,

bar(**c**, 0, 600000, 20) **AS** bar

**FROM** test.hits

**GROUP** **BY** h

**ORDER** **BY** h **ASC**

┌──h─┬──────c─┬─bar────────────────┐

│ 0 │ 292907 │ █████████▋ │

│ 1 │ 180563 │ ██████ │

│ 2 │ 114861 │ ███▋ │

│ 3 │ 85069 │ ██▋ │

│ 4 │ 68543 │ ██▎ │

│ 5 │ 78116 │ ██▌ │

│ 6 │ 113474 │ ███▋ │

│ 7 │ 170678 │ █████▋ │

│ 8 │ 278380 │ █████████▎ │

│ 9 │ 391053 │ █████████████ │

│ 10 │ 457681 │ ███████████████▎ │

│ 11 │ 493667 │ ████████████████▍ │

│ 12 │ 509641 │ ████████████████▊ │

│ 13 │ 522947 │ █████████████████▍ │

│ 14 │ 539954 │ █████████████████▊ │

│ 15 │ 528460 │ █████████████████▌ │

│ 16 │ 539201 │ █████████████████▊ │

│ 17 │ 523539 │ █████████████████▍ │

│ 18 │ 506467 │ ████████████████▊ │

│ 19 │ 520915 │ █████████████████▎ │

│ 20 │ 521665 │ █████████████████▍ │

│ 21 │ 542078 │ ██████████████████ │

│ 22 │ 493642 │ ████████████████▍ │

│ 23 │ 400397 │ █████████████▎ │

└────┴────────┴────────────────────┘

###### transform

Преобразовать значение согласно явно указанному отображению, одних элементов на другие.

Имеется два варианта функции:

transform(x, array\_from, array\_to, default)

x – что преобразовывать.

array\_from – константный массив значений для преобразования.

array\_to – константный массив значений, в которые должны быть преобразованы значения из from.

default – какое значение использовать, если x не равен ни одному из значений во from.

array\_from и array\_to – массивы одинаковых размеров.

Типы:

transform(T, Array(T), Array(U), U) -> U

T и U – могут быть числовыми, строковыми, или Date или DateTime типами.

При этом, где обозначена одна и та же буква (T или U), могут быть, в случае числовых типов, не совпадающие типы, а типы, для которых есть общий тип.

Например, первый аргумент может иметь тип Int64, а второй – Array(UInt16).

Если значение x равно одному из элементов массива array\_from, то возвращает соответствующий (такой же по номеру) элемент массива array\_to; иначе возвращает default. Если имеется несколько совпадающих элементов в array\_from, то возвращает какой-нибудь из соответствующих.

Пример:

**SELECT**

**transform**(SearchEngineID, [2, 3], ['Yandex', 'Google'], 'Other') **AS** title,

**count**() **AS** **c**

**FROM** test.hits

**WHERE** SearchEngineID != 0

**GROUP** **BY** title

**ORDER** **BY** **c** **DESC**

┌─title─────┬──────c─┐

│ Yandex │ 498635 │

│ Google │ 229872 │

│ Other │ 104472 │

└───────────┴────────┘

transform(x, array\_from, array\_to)

Отличается от первого варианта отсутствующим аргументом default.

Если значение x равно одному из элементов массива array\_from, то возвращает соответствующий (такой же по номеру) элемент массива array\_to; иначе возвращает x.

Типы:

transform(T, Array(T), Array(T)) -> T

Пример:

**SELECT**

**transform**(**domain**(Referer), ['yandex.ru', 'google.ru', 'vk.com'], ['www.yandex', 'example.com']) **AS** s,

**count**() **AS** **c**

**FROM** test.hits

**GROUP** **BY** **domain**(Referer)

**ORDER** **BY** **count**() **DESC**

**LIMIT** 10

┌─s──────────────┬───────c─┐

│ │ 2906259 │

│ www.yandex │ 867767 │

│ ███████.ru │ 313599 │

│ mail.yandex.ru │ 107147 │

│ ██████.ru │ 100355 │

│ █████████.ru │ 65040 │

│ news.yandex.ru │ 64515 │

│ ██████.net │ 59141 │

│ example.com │ 57316 │

└────────────────┴─────────┘

###### formatReadableSize(x)

Принимает размер (число байт). Возвращает округленный размер с суффиксом (KiB, MiB и т.д.) в виде строки.

Пример:

**SELECT**

arrayJoin([1, 1024, 1024\*1024, 192851925]) **AS** filesize\_bytes,

formatReadableSize(filesize\_bytes) AS filesize

┌─filesize\_bytes─┬─filesize───┐

│ 1 │ 1.00 B │

│ 1024 │ 1.00 KiB │

│ 1048576 │ 1.00 MiB │

│ 192851925 │ 183.92 MiB │

└────────────────┴────────────┘

###### formatReadableQuantity(x)

Принимает число. Возвращает округленное число с суффиксом (thousand, million, billion и т.д.) в виде строки.

Облегчает визуальное восприятие больших чисел живым человеком.

Пример:

**SELECT**

arrayJoin([1024, 1234 \* 1000, (4567 \* 1000) \* 1000, 98765432101234]) **AS** number,

formatReadableQuantity(number) **AS** number\_for\_humans

┌─────────number─┬─number\_for\_humans─┐

│ 1024 │ 1.02 thousand │

│ 1234000 │ 1.23 million │

│ 4567000000 │ 4.57 billion │

│ 98765432101234 │ 98.77 trillion │

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

###### least(a, b)

Возвращает наименьшее значение из a и b.

###### greatest(a, b)

Возвращает наибольшее значение из a и b.

###### uptime()

Возвращает аптайм сервера в секундах.

###### version()

Возвращает версию сервера в виде строки.

###### rowNumberInBlock

Возвращает порядковый номер строки в блоке данных. Для каждого блока данных нумерация начинается с 0.

###### rowNumberInAllBlocks()

Возвращает порядковый номер строки в блоке данных. Функция учитывает только задействованные блоки данных.

###### neighbor

Функция позволяет получить доступ к значению в столбце column, находящемуся на смещении offset относительно текущей строки. Является частичной реализацией [оконных функций](https://en.wikipedia.org/wiki/SQL_window_function) LEAD() и LAG().

Синтаксис:

neighbor(**column**, **offset**[, default\_value])

Результат функции зависит от затронутых блоков данных и порядка данных в блоке.

***Предупреждение:*** *Функция может получить доступ к значению в столбце соседней строки только внутри обрабатываемого в данный момент блока данных.*

Порядок строк, используемый при вычислении функции neighbor, может отличаться от порядка строк, возвращаемых пользователю.

Чтобы этого не случилось, вы можете сделать подзапрос с [ORDER BY](https://clickhouse.tech/docs/ru/sql-reference/statements/select/order-by/) и вызвать функцию изне подзапроса.

Параметры:

* column – имя столбца или скалярное выражение.
* offset – смещение от текущей строки column. [Int64](https://clickhouse.tech/docs/ru/sql-reference/functions/other-functions/).
* default\_value – опциональный параметр. Значение, которое будет возвращено, если смещение выходит за пределы блока данных.

Возвращаемое значение:

* Значение column в смещении от текущей строки, если значение offset не выходит за пределы блока.
* Значение по умолчанию для column, если значение offset выходит за пределы блока данных. Если передан параметр default\_value, то значение берется из него.

Тип: зависит от данных в column или переданного значения по умолчанию в default\_value.

Пример:

Запрос:

**SELECT** number, neighbor(number, 2) **FROM** **system**.numbers **LIMIT** 10;

Ответ:

┌─number─┬─neighbor(number, 2)─┐

│ 0 │ 2 │

│ 1 │ 3 │

│ 2 │ 4 │

│ 3 │ 5 │

│ 4 │ 6 │

│ 5 │ 7 │

│ 6 │ 8 │

│ 7 │ 9 │

│ 8 │ 0 │

│ 9 │ 0 │

└────────┴─────────────────────┘

Запрос:

**SELECT** number, neighbor(number, 2, 999) **FROM** **system**.numbers **LIMIT** 10;

Ответ:

┌─number─┬─neighbor(number, 2, 999)─┐

│ 0 │ 2 │

│ 1 │ 3 │

│ 2 │ 4 │

│ 3 │ 5 │

│ 4 │ 6 │

│ 5 │ 7 │

│ 6 │ 8 │

│ 7 │ 9 │

│ 8 │ 999 │

│ 9 │ 999 │

└────────┴──────────────────────────┘

Эта функция может использоваться для оценки year-over-year значение показателя:

Запрос:

**WITH** toDate('2018-01-01') **AS** start\_date

**SELECT**

toStartOfMonth(start\_date + (number \* 32)) **AS** **month**,

toInt32(**month**) % 100 **AS** money,

neighbor(money, -12) **AS** prev\_year,

round(prev\_year / money, 2) **AS** year\_over\_year

**FROM** numbers(16)

Ответ:

┌──────month─┬─money─┬─prev\_year─┬─year\_over\_year─┐

│ 2018-01-01 │ 32 │ 0 │ 0 │

│ 2018-02-01 │ 63 │ 0 │ 0 │

│ 2018-03-01 │ 91 │ 0 │ 0 │

│ 2018-04-01 │ 22 │ 0 │ 0 │

│ 2018-05-01 │ 52 │ 0 │ 0 │

│ 2018-06-01 │ 83 │ 0 │ 0 │

│ 2018-07-01 │ 13 │ 0 │ 0 │

│ 2018-08-01 │ 44 │ 0 │ 0 │

│ 2018-09-01 │ 75 │ 0 │ 0 │

│ 2018-10-01 │ 5 │ 0 │ 0 │

│ 2018-11-01 │ 36 │ 0 │ 0 │

│ 2018-12-01 │ 66 │ 0 │ 0 │

│ 2019-01-01 │ 97 │ 32 │ 0.33 │

│ 2019-02-01 │ 28 │ 63 │ 2.25 │

│ 2019-03-01 │ 56 │ 91 │ 1.62 │

│ 2019-04-01 │ 87 │ 22 │ 0.25 │

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

###### runningDifference(x)

Считает разницу между последовательными значениями строк в блоке данных.

Возвращает 0 для первой строки и разницу с предыдущей строкой для каждой последующей строки.

***Предупреждение:*** *Функция может взять значение предыдущей строки только внутри текущего обработанного блока данных.*

Результат функции зависит от затронутых блоков данных и порядка данных в блоке.

Порядок строк, используемый при вычислении функции runningDifference, может отличаться от порядка строк, возвращаемых пользователю.

Чтобы этого не случилось, вы можете сделать подзапрос с [ORDER BY](https://clickhouse.tech/docs/ru/sql-reference/statements/select/order-by/) и вызвать функцию извне подзапроса.

Пример:

**SELECT**

EventID,

EventTime,

runningDifference(EventTime) **AS** delta

**FROM**

(

**SELECT**

EventID,

EventTime

**FROM** events

**WHERE** EventDate = '2016-11-24'

**ORDER** **BY** EventTime **ASC**

**LIMIT** 5

)

┌─EventID─┬───────────EventTime─┬─delta─┐

│ 1106 │ 2016-11-24 00:00:04 │ 0 │

│ 1107 │ 2016-11-24 00:00:05 │ 1 │

│ 1108 │ 2016-11-24 00:00:05 │ 0 │

│ 1109 │ 2016-11-24 00:00:09 │ 4 │

│ 1110 │ 2016-11-24 00:00:10 │ 1 │

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

Обратите внимание – размер блока влияет на результат. С каждым новым блоком состояние runningDifference сбрасывается:

**SELECT**

number,

runningDifference(number + 1) **AS** diff

**FROM** numbers(100000)

**WHERE** diff != 1

┌─number─┬─diff─┐

│ 0 │ 0 │

└────────┴──────┘

┌─number─┬─diff─┐

│ 65536 │ 0 │

└────────┴──────┘

**set** max\_block\_size=100000 -- по умолчанию 65536!

**SELECT**

number,

runningDifference(number + 1) **AS** diff

**FROM** numbers(100000)

**WHERE** diff != 1

┌─number─┬─diff─┐

│ 0 │ 0 │

└────────┴──────┘

###### runningDifferenceStartingWithFirstValue

То же, что и [runningDifference] (./other\_functions.md # other\_functions-runningdifference), но в первой строке возвращается значение первой строки, а не ноль.

###### MACNumToString(num)

Принимает число типа UInt64. Интерпретирует его, как MAC-адрес в big endian. Возвращает строку, содержащую соответствующий MAC-адрес в формате AA:BB:CC:DD:EE:FF (числа в шестнадцатеричной форме через двоеточие).

###### MACStringToNum(s)

Функция, обратная к MACNumToString. Если MAC адрес в неправильном формате, то возвращает 0.

###### MACStringToOUI(s)

Принимает MAC адрес в формате AA:BB:CC:DD:EE:FF (числа в шестнадцатеричной форме через двоеточие). Возвращает первые три октета как число в формате UInt64. Если MAC адрес в неправильном формате, то возвращает 0.

###### getSizeOfEnumType

Возвращает количество полей в [Enum](https://clickhouse.tech/docs/ru/sql-reference/functions/other-functions/).

Синтаксис:

getSizeOfEnumType(value)

Параметры:

* value – значение типа Enum.

Возвращаемые значения:

* Количество полей входного значения типа Enum.
* Исключение, если тип не Enum.

Пример:

**SELECT** getSizeOfEnumType( **CAST**('a' **AS** Enum8('a' = 1, 'b' = 2) ) ) **AS** x

┌─x─┐

│ 2 │

└───┘

###### blockSerializedSize

Возвращает размер на диске (без учета сжатия).

Синтаксис:

blockSerializedSize(value[, value[, ...]])

Параметры:

* value – значение произвольного типа.

Возвращаемые значения:

* Количество байтов, которые будут записаны на диск для блока значений (без сжатия).

Пример:

Запрос:

**SELECT** blockSerializedSize(maxState(1)) **as** x

Ответ:

┌─x─┐

│ 2 │

└───┘

###### toColumnTypeName

Возвращает имя класса, которым представлен тип данных столбца в оперативной памяти.

Синтаксис:

toColumnTypeName(value)

Параметры:

* value – значение произвольного типа.

Возвращаемые значения:

* Строка с именем класса, который используется для представления типа данных value в оперативной памяти.

Пример разницы между toTypeName и toColumnTypeName:

**SELECT** toTypeName(**CAST**('2018-01-01 01:02:03' **AS** DateTime))

┌─toTypeName(CAST('2018-01-01 01:02:03', 'DateTime'))─┐

│ DateTime │

└─────────────────────────────────────────────────────┘

**SELECT** toColumnTypeName(**CAST**('2018-01-01 01:02:03' **AS** DateTime))

┌─toColumnTypeName(CAST('2018-01-01 01:02:03', 'DateTime'))─┐

│ Const(UInt32) │

└───────────────────────────────────────────────────────────┘

В примере видно, что тип данных DateTime хранится в памяти как Const(UInt32).

###### dumpColumnStructure

Выводит развернутое описание структур данных в оперативной памяти.

Синтаксис:

dumpColumnStructure(value)

Параметры:

* value – значение произвольного типа.

Возвращаемые значения:

* Строка с описанием структуры, которая используется для представления типа данных value в оперативной памяти.

Пример:

**SELECT** dumpColumnStructure(**CAST**('2018-01-01 01:02:03', 'DateTime'))

┌─dumpColumnStructure(CAST('2018-01-01 01:02:03', 'DateTime'))─┐

│ DateTime, Const(size = 1, UInt32(size = 1)) │

└──────────────────────────────────────────────────────────────┘

###### defaultValueOfArgumentType

Выводит значение по умолчанию для типа данных.

Не учитывает значения по умолчанию для столбцов, заданные пользователем.

Синтаксис:

defaultValueOfArgumentType(expression)

Параметры:

* expression – значение произвольного типа или выражение, результатом которого является значение произвольного типа.

Возвращаемые значения:

* 0 для чисел;
* Пустая строка для строк;
* ᴺᵁᴸᴸ для [Nullable](https://clickhouse.tech/docs/ru/sql-reference/functions/other-functions/).

Пример:

**SELECT** defaultValueOfArgumentType( **CAST**(1 **AS** Int8) )

┌─defaultValueOfArgumentType(CAST(1, 'Int8'))─┐

│ 0 │

└─────────────────────────────────────────────┘

**SELECT** defaultValueOfArgumentType( **CAST**(1 **AS** **Nullable**(Int8) ) )

┌─defaultValueOfArgumentType(CAST(1, 'Nullable(Int8)'))─┐

│ ᴺᵁᴸᴸ │

└───────────────────────────────────────────────────────┘

###### defaultValueOfTypeName

Выводит значение по умолчанию для указанного типа данных.

Не включает значения по умолчанию для настраиваемых столбцов, установленных пользователем.

Синтаксис:

defaultValueOfTypeName(**type**)

Параметры:

* type – тип данных.

Возвращаемое значение:

* 0 для чисел;
* Пустая строка для строк;
* ᴺᵁᴸᴸ для [Nullable](https://clickhouse.tech/docs/ru/sql-reference/data-types/nullable/).

Пример:

**SELECT** defaultValueOfTypeName('Int8')

┌─defaultValueOfTypeName('Int8')─┐

│ 0 │

└────────────────────────────────┘

**SELECT** defaultValueOfTypeName('Nullable(Int8)')

┌─defaultValueOfTypeName('Nullable(Int8)')─┐

│ ᴺᵁᴸᴸ │

└──────────────────────────────────────────┘

###### replicate

Создает массив, заполненный одним значением.

Используется для внутренней реализации [arrayJoin](https://clickhouse.tech/docs/ru/sql-reference/functions/array-join/#functions_arrayjoin).

Синтаксис:

**SELECT** replicate(x, arr);

Параметры:

* arr – исходный массив. RT.WideStore создаёт новый массив такой же длины как исходный и заполняет его значением x.
* x – значение, которым будет заполнен результирующий массив.

Возвращаемое значение:

* Массив, заполненный значением x.

Тип: Array.

Пример:

Запрос:

**SELECT** replicate(1, ['a', 'b', 'c']);

Ответ:

┌─replicate(1, ['a', 'b', 'c'])─┐

│ [1,1,1] │

└───────────────────────────────┘

###### filesystemAvailable

Возвращает объём доступного для записи данных места на файловой системе. Он всегда меньше общего свободного места ([filesystemFree](https://clickhouse.tech/docs/ru/sql-reference/functions/other-functions/#filesystemfree)), потому что некоторое пространство зарезервировано для нужд операционной системы.

Синтаксис:

filesystemAvailable()

Возвращаемое значение:

* Объём доступного для записи данных места в байтах.

Тип: [UInt64](https://clickhouse.tech/docs/ru/sql-reference/functions/other-functions/).

Пример:

Запрос:

**SELECT** formatReadableSize(filesystemAvailable()) **AS** "Available space", toTypeName(filesystemAvailable()) **AS** "Type";

Ответ:

┌─Available space─┬─Type───┐

│ 30.75 GiB │ UInt64 │

└─────────────────┴────────┘

###### filesystemFree

Возвращает объём свободного места на файловой системе. Смотрите также filesystemAvailable.

Синтаксис:

filesystemFree()

Возвращаемое значение:

* Объем свободного места в байтах.

Тип: [UInt64](https://clickhouse.tech/docs/ru/sql-reference/functions/other-functions/).

Пример:

Запрос:

**SELECT** formatReadableSize(filesystemFree()) **AS** "Free space", toTypeName(filesystemFree()) **AS** "Type";

Ответ:

┌─Free space─┬─Type───┐

│ 32.39 GiB │ UInt64 │

└────────────┴────────┘

###### filesystemCapacity

Возвращает информацию о ёмкости файловой системы в байтах. Для оценки должен быть настроен [путь](https://clickhouse.tech/docs/ru/sql-reference/functions/other-functions/#server_configuration_parameters-path) к каталогу с данными.

Синтаксис:

filesystemCapacity()

Возвращаемое значение:

* Информация о ёмкости файловой системы в байтах.

Тип: [UInt64](https://clickhouse.tech/docs/ru/sql-reference/functions/other-functions/).

Пример:

Запрос:

**SELECT** formatReadableSize(filesystemCapacity()) **AS** "Capacity", toTypeName(filesystemCapacity()) **AS** "Type"

Ответ:

┌─Capacity──┬─Type───┐

│ 39.32 GiB │ UInt64 │

└───────────┴────────┘

###### finalizeAggregation

Принимает состояние агрегатной функции. Возвращает результат агрегирования (или конечное состояние при использовании комбинатора [-State](https://clickhouse.tech/docs/ru/sql-reference/aggregate-functions/combinators/#state)).

Синтаксис:

finalizeAggregation(**state**)

Параметры:

* state – состояние агрегатной функции. [AggregateFunction](https://clickhouse.tech/docs/ru/sql-reference/data-types/aggregatefunction/#data-type-aggregatefunction).

Возвращаемые значения:

* Значения, которые были агрегированы.

Тип: соответствует типу агрегируемых значений.

Примеры:

Запрос:

**SELECT** finalizeAggregation(( **SELECT** countState(number) **FROM** numbers(10)));

Результат:

┌─finalizeAggregation(\_subquery16)─┐

│ 10 │

└──────────────────────────────────┘

Запрос:

**SELECT** finalizeAggregation(( **SELECT** sumState(number) **FROM** numbers(10)));

Результат:

┌─finalizeAggregation(\_subquery20)─┐

│ 45 │

└──────────────────────────────────┘

Обратите внимание, что значения NULL игнорируются.

Запрос:

**SELECT** finalizeAggregation(arrayReduce('anyState', [**NULL**, 2, 3]));

Результат:

┌─finalizeAggregation(arrayReduce('anyState', [NULL, 2, 3]))─┐

│ 2 │

└────────────────────────────────────────────────────────────┘

Комбинированный пример:

Запрос:

**WITH** initializeAggregation('sumState', number) **AS** one\_row\_sum\_state

**SELECT**

number,

finalizeAggregation(one\_row\_sum\_state) **AS** one\_row\_sum,

runningAccumulate(one\_row\_sum\_state) **AS** cumulative\_sum

**FROM** numbers(10);

Результат:

┌─number─┬─one\_row\_sum─┬─cumulative\_sum─┐

│ 0 │ 0 │ 0 │

│ 1 │ 1 │ 1 │

│ 2 │ 2 │ 3 │

│ 3 │ 3 │ 6 │

│ 4 │ 4 │ 10 │

│ 5 │ 5 │ 15 │

│ 6 │ 6 │ 21 │

│ 7 │ 7 │ 28 │

│ 8 │ 8 │ 36 │

│ 9 │ 9 │ 45 │

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

Смотрите также:

* [arrayReduce](https://clickhouse.tech/docs/ru/sql-reference/functions/array-functions/#arrayreduce),
* [initializeAggregation](https://clickhouse.tech/docs/ru/sql-reference/aggregate-functions/reference/initializeAggregation/).

###### runningAccumulate

Накапливает состояния агрегатной функции для каждой строки блока данных.

Функция обнуляет состояние для каждого нового блока.

Синтаксис:

runningAccumulate(agg\_state[, **grouping**]);

Параметры:

* agg\_state – состояние агрегатной функции. [AggregateFunction](https://clickhouse.tech/docs/ru/sql-reference/data-types/aggregatefunction/#data-type-aggregatefunction).
* grouping – ключ группировки. Опциональный параметр. Состояние функции обнуляется, если значение grouping меняется. Параметр может быть любого [поддерживаемого типа данных](https://clickhouse.tech/docs/ru/sql-reference/data-types/), для которого определен оператор равенства.

Возвращаемое значение:

* Каждая результирующая строка содержит результат агрегатной функции, накопленный для всех входных строк от 0 до текущей позиции. runningAccumulate обнуляет состояния для каждого нового блока данных или при изменении значения grouping.

Тип зависит от используемой агрегатной функции.

Примеры:

Рассмотрим примеры использования runningAccumulate для нахождения кумулятивной суммы чисел без и с группировкой.

Запрос:

**SELECT** k, runningAccumulate(sum\_k) **AS** res **FROM** (**SELECT** number **as** k, sumState(k) **AS** sum\_k **FROM** numbers(10) **GROUP** **BY** k **ORDER** **BY** k);

Результат:

┌─k─┬─res─┐

│ 0 │ 0 │

│ 1 │ 1 │

│ 2 │ 3 │

│ 3 │ 6 │

│ 4 │ 10 │

│ 5 │ 15 │

│ 6 │ 21 │

│ 7 │ 28 │

│ 8 │ 36 │

│ 9 │ 45 │

└───┴─────┘

Подзапрос формирует sumState для каждого числа от 0 до 9. sumState возвращает состояние функции [sum](https://clickhouse.tech/docs/ru/sql-reference/aggregate-functions/reference/sum/#agg_function-sum), содержащее сумму одного числа.

Весь запрос делает следующее:

1. Для первой строки runningAccumulate берет sumState(0) и возвращает 0.
2. Для второй строки функция объединяет sumState (0) и sumState (1), что приводит к sumState (0 + 1), и возвращает в результате 1.
3. Для третьей строки функция объединяет sumState (0 + 1) и sumState (2), что приводит к sumState (0 + 1 + 2), и в результате возвращает 3.
4. Действия повторяются до тех пор, пока не закончится блок.

В следующем примере показано использование параметра grouping:

Запрос:

**SELECT**

**grouping**,

item,

runningAccumulate(state, **grouping**) **AS** res

**FROM**

(

**SELECT**

toInt8(number / 4) **AS** **grouping**,

number **AS** item,

sumState(number) **AS** **state**

**FROM** numbers(15)

**GROUP** **BY** item

**ORDER** **BY** item **ASC**

);

Результат:

┌─grouping─┬─item─┬─res─┐

│ 0 │ 0 │ 0 │

│ 0 │ 1 │ 1 │

│ 0 │ 2 │ 3 │

│ 0 │ 3 │ 6 │

│ 1 │ 4 │ 4 │

│ 1 │ 5 │ 9 │

│ 1 │ 6 │ 15 │

│ 1 │ 7 │ 22 │

│ 2 │ 8 │ 8 │

│ 2 │ 9 │ 17 │

│ 2 │ 10 │ 27 │

│ 2 │ 11 │ 38 │

│ 3 │ 12 │ 12 │

│ 3 │ 13 │ 25 │

│ 3 │ 14 │ 39 │

└──────────┴──────┴─────┘

Как вы можете видеть, runningAccumulate объединяет состояния для каждой группы строк отдельно.

###### joinGet

Функция позволяет извлекать данные из таблицы таким же образом как из [словаря](https://clickhouse.tech/docs/ru/sql-reference/functions/other-functions/).

Получает данные из таблиц [Join](https://clickhouse.tech/docs/ru/sql-reference/functions/other-functions/#creating-a-table) по ключу.

Поддерживаются только таблицы, созданные с ENGINE = Join(ANY, LEFT, <join\_keys>).

Синтаксис:

joinGet(join\_storage\_table\_name, `value\_column`, join\_keys)

Параметры:

* join\_storage\_table\_name – [идентификатор](https://clickhouse.tech/docs/ru/sql-reference/syntax/#syntax-identifiers), который указывает, откуда производится выборка данных. Поиск по идентификатору осуществляется в базе данных по умолчанию (см. конфигурацию default\_database). Чтобы переопределить базу данных по умолчанию, используйте команду USE db\_name, или укажите базу данных и таблицу через разделитель db\_name.db\_table, см. пример.
* value\_column – столбец, из которого нужно произвести выборку данных.
* join\_keys – список ключей, по которым производится выборка данных.

Возвращаемое значение:

* Возвращает значение по списку ключей.

Если значения не существует в исходной таблице, вернется 0 или null в соответствии с настройками [join\_use\_nulls](https://clickhouse.tech/docs/ru/operations/settings/settings/#join_use_nulls).

Подробнее о настройке join\_use\_nulls в [операциях Join](https://clickhouse.tech/docs/ru/sql-reference/functions/other-functions/).

Пример:

Входная таблица:

**CREATE** **DATABASE** db\_test

**CREATE** **TABLE** db\_test.id\_val(`id` UInt32, `val` UInt32) ENGINE = **Join**(**ANY**, **LEFT**, id) SETTINGS join\_use\_nulls = 1

**INSERT** **INTO** db\_test.id\_val **VALUES** (1,11)(2,12)(4,13)

┌─id─┬─val─┐

│ 4 │ 13 │

│ 2 │ 12 │

│ 1 │ 11 │

└────┴─────┘

Запрос:

**SELECT** joinGet(db\_test.id\_val,'val',toUInt32(number)) **from** numbers(4) SETTINGS join\_use\_nulls = 1

Результат:

┌─joinGet(db\_test.id\_val, 'val', toUInt32(number))─┐

│ 0 │

│ 11 │

│ 12 │

│ 0 │

└──────────────────────────────────────────────────┘

###### modelEvaluate(model\_name, …)

Оценивает внешнюю модель.

Принимает на вход имя и аргументы модели. Возвращает Float64.

###### throwIf(x[, custom\_message])

Бросает исключение, если аргумент не равен нулю.

custom\_message – необязательный параметр, константная строка, задает текст сообщения об ошибке.

**SELECT** throwIf(number = 3, 'Too many') **FROM** numbers(10);

↙ Progress: 0.00 rows, 0.00 B (0.00 rows/s., 0.00 B/s.) Received exception from server (version 19.14.1):

Code: 395. DB::Exception: Received from localhost:9000. DB::Exception: Too many.

###### identity

Возвращает свой аргумент. Используется для отладки и тестирования, позволяет отменить использование индекса, и получить результат и производительность полного сканирования таблицы. Это работает, потому что оптимизатор запросов не может «заглянуть» внутрь функции identity.

Синтаксис:

**identity**(x)

Пример:

Query:

**SELECT** **identity**(42)

Результат:

┌─identity(42)─┐

│ 42 │

└──────────────┘

###### randomPrintableASCII

Генерирует строку со случайным набором печатных символов [ASCII](https://en.wikipedia.org/wiki/ASCII#Printable_characters).

Синтаксис:

randomPrintableASCII(**length**)

Параметры:

* length – Длина результирующей строки. Положительное целое число.

Если передать `length < 0`, то поведение функции не определено.

Возвращаемое значение:

* Строка со случайным набором печатных символов [ASCII](https://en.wikipedia.org/wiki/ASCII#Printable_characters).

Тип: [String](https://clickhouse.tech/docs/ru/sql-reference/functions/other-functions/).

Пример:

**SELECT** number, randomPrintableASCII(30) **as** str, **length**(str) **FROM** **system**.numbers **LIMIT** 3

┌─number─┬─str────────────────────────────┬─length(randomPrintableASCII(30))─┐

│ 0 │ SuiCOSTvC0csfABSw=UcSzp2.`rv8x │ 30 │

│ 1 │ 1Ag NlJ &RCN:\*>HVPG;PE-nO"SUFD │ 30 │

│ 2 │ /"+<"wUTh:=LjJ Vm!c&hI\*m#XTfzz │ 30 │

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

###### randomString

Генерирует бинарную строку заданной длины, заполненную случайными байтами (в том числе нулевыми).

Синтаксис:

randomString(**length**)

Параметры:

* length – длина строки. Положительное целое число.

Возвращаемое значение:

* Строка, заполненная случайными байтами.

Type: [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).

Пример:

Запрос:

**SELECT** randomString(30) **AS** str, **length**(str) **AS** len **FROM** numbers(2) FORMAT Vertical;

Ответ:

Row 1:

──────

str: 3 G : pT ?w тi k aV f6

len: 30

Row 2:

──────

str: 9 ,] ^ ) ]?? 8

len: 30

Смотрите также:

* [generateRandom](https://clickhouse.tech/docs/ru/sql-reference/table-functions/generate/#generaterandom),
* [randomPrintableASCII](https://clickhouse.tech/docs/ru/sql-reference/functions/other-functions/#randomascii).

###### randomFixedString

Генерирует бинарную строку заданной длины, заполненную случайными байтами, включая нулевые.

Синтаксис:

randomFixedString(**length**);

Параметры:

* length – длина строки в байтах. [UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Возвращаемое значение:

* Строка, заполненная случайными байтами.

Тип: [FixedString](https://clickhouse.tech/docs/ru/sql-reference/data-types/fixedstring/).

Пример:

Запрос:

**SELECT** randomFixedString(13) **as** rnd, toTypeName(rnd)

Результат:

┌─rnd──────┬─toTypeName(randomFixedString(13))─┐

│ j▒h㋖HɨZ'▒ │ FixedString(13) │

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

###### randomStringUTF8

Генерирует строку заданной длины со случайными символами в кодировке UTF-8.

Синтаксис:

randomStringUTF8(**length**);

Параметры:

* length – длина итоговой строки в кодовых точках. [UInt64](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Возвращаемое значение:

* Случайная строка в кодировке UTF-8.

Тип: [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).

Пример:

Запрос:

**SELECT** randomStringUTF8(13)

Результат:

┌─randomStringUTF8(13)─┐

│ 𘤗𙉝д兠庇󡅴󱱎󦐪􂕌𔊹𓰛 │

└──────────────────────┘

###### getSetting

Возвращает текущее значение [пользовательской настройки](https://clickhouse.tech/docs/ru/operations/settings/#custom_settings).

Синтаксис:

getSetting('custom\_setting');

Параметр:

* custom\_setting – название настройки. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/).

Возвращаемое значение

* Текущее значение пользовательской настройки.

Пример:

**SET** custom\_a = 123;

**SELECT** getSetting('custom\_a');

Результат:

123

См. также:

* [Пользовательские настройки](https://clickhouse.tech/docs/ru/operations/settings/#custom_settings).

###### isDecimalOverflow

Проверяет, находится ли число [Decimal](https://clickhouse.tech/docs/ru/sql-reference/data-types/decimal/) вне собственной (или заданной) области значений.

Синтаксис:

isDecimalOverflow(d, [p])

Параметры:

* d – число. [Decimal](https://clickhouse.tech/docs/ru/sql-reference/data-types/decimal/).
* p – точность. Необязательный параметр. Если опущен, используется исходная точность первого аргумента. Использование этого параметра может быть полезно для извлечения данных в другую СУБД или файл. [UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges).

Возвращаемое значение:

* 1 – число имеет больше цифр, чем позволяет точность.
* 0 – число удовлетворяет заданной точности.

Пример:

Запрос:

**SELECT** isDecimalOverflow(toDecimal32(1000000000, 0), 9),

isDecimalOverflow(toDecimal32(1000000000, 0)),

isDecimalOverflow(toDecimal32(-1000000000, 0), 9),

isDecimalOverflow(toDecimal32(-1000000000, 0));

Результат:

1 1 1 1

###### countDigits

Возвращает количество десятичных цифр, необходимых для представления значения.

Синтаксис:

countDigits(x)

Параметры:

* x – [целое](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint8-uint16-uint32-uint64-int8-int16-int32-int64) или [дробное](https://clickhouse.tech/docs/ru/sql-reference/data-types/decimal/) число.

Возвращаемое значение:

* Количество цифр.

Тип: [UInt8](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/#uint-ranges).

Для Decimal значений учитывается их масштаб: вычисляется результат по базовому целочисленному типу, полученному как (value \* scale). Например: countDigits(42) = 2, countDigits(42.000) = 5, countDigits(0.04200) = 4. То есть вы можете проверить десятичное переполнение для Decimal64 с помощью countDecimal(x) > 18. Это медленный вариант [isDecimalOverflow](https://clickhouse.tech/docs/ru/sql-reference/functions/other-functions/#is-decimal-overflow).

Пример:

Запрос:

**SELECT** countDigits(toDecimal32(1, 9)), countDigits(toDecimal32(-1, 9)),

countDigits(toDecimal64(1, 18)), countDigits(toDecimal64(-1, 18)),

countDigits(toDecimal128(1, 38)), countDigits(toDecimal128(-1, 38));

Результат:

10 10 19 19 39 39

###### errorCodeToName

Возвращаемое значение:

* Название переменной для кода ошибки.

Тип: [LowCardinality(String)](https://clickhouse.tech/docs/ru/sql-reference/data-types/lowcardinality/).

Синтаксис:

errorCodeToName(1)

Результат:

UNSUPPORTED\_METHOD

###### tcpPort

Вовращает номер TCP порта, который использует сервер для [нативного протокола](https://clickhouse.tech/docs/ru/interfaces/tcp/).

Синтаксис:

tcpPort()

Параметры:

* Нет.

Возвращаемое значение:

* Номер TCP порта.

Тип: [UInt16](https://clickhouse.tech/docs/ru/sql-reference/data-types/int-uint/).

Пример:

Запрос:

**SELECT** tcpPort();

Результат:

┌─tcpPort()─┐

│ 9000 │

└───────────┘

Смотрите также:

* [tcp\_port](https://clickhouse.tech/docs/ru/operations/server-configuration-parameters/settings/#server_configuration_parameters-tcp_port).

##### Функции шифрования

Даннвые функции реализуют шифрование и расшифровку данных с помощью AES (Advanced Encryption Standard) алгоритма.

Длина ключа зависит от режима шифрования. Он может быть длинной в 16, 24 и 32 байта для режимов шифрования -128-, -196- и -256- соответственно.

Длина инициализирующего вектора всегда 16 байт (лишнии байты игнорируются).

Обратите внимание, что до версии RT.WideStore 21.1 эти функции работали медленно.

###### encrypt

Функция поддерживает шифрование данных следующими режимами:

* aes-128-ecb, aes-192-ecb, aes-256-ecb
* aes-128-cbc, aes-192-cbc, aes-256-cbc
* aes-128-cfb1, aes-192-cfb1, aes-256-cfb1
* aes-128-cfb8, aes-192-cfb8, aes-256-cfb8
* aes-128-cfb128, aes-192-cfb128, aes-256-cfb128
* aes-128-ofb, aes-192-ofb, aes-256-ofb
* aes-128-gcm, aes-192-gcm, aes-256-gcm

Синтаксис:

encrypt('mode', 'plaintext', 'key' [, iv, aad])

Параметры:

* mode – режим шифрования. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/#string).
* plaintext – текст, который будет зашифрован. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/#string).
* key – ключ шифрования. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/#string).
* iv – инициализирующий вектор. Обязателен для -gcm режимов, для остальных режимов необязателен. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/#string).
* aad – дополнительные аутентифицированные данные. Не шифруются, но влияют на расшифровку. Параметр работает только с -gcm режимами. Для остальных вызовет исключение. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/#string).

Возвращаемое значение:

* Бинарная зашифрованная строка. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/#string).

Примеры:

Создадим такую таблицу:

Запрос:

**CREATE** **TABLE** encryption\_test

(

`**comment**` String,

`secret` String

)

ENGINE = Memory;

Вставим некоторые данные (замечание: не храните ключи или инициализирующие векторы в базе данных, так как это компрометирует всю концепцию шифрования), также хранение "подсказок" небезопасно и используется только для наглядности:

Запрос:

**INSERT** **INTO** encryption\_test **VALUES**('aes-256-cfb128 no IV', encrypt('aes-256-cfb128', 'Secret', '12345678910121314151617181920212')),\

('aes-256-cfb128 no IV, different key', encrypt('aes-256-cfb128', 'Secret', 'keykeykeykeykeykeykeykeykeykeyke')),\

('aes-256-cfb128 with IV', encrypt('aes-256-cfb128', 'Secret', '12345678910121314151617181920212', 'iviviviviviviviv')),\

('aes-256-cbc no IV', encrypt('aes-256-cbc', 'Secret', '12345678910121314151617181920212'));

Запрос:

**SELECT** **comment**, hex(secret) **FROM** encryption\_test;

Результат:

┌─comment─────────────────────────────┬─hex(secret)──────────────────────┐

│ aes-256-cfb128 no IV │ B4972BDC4459 │

│ aes-256-cfb128 no IV, different key │ 2FF57C092DC9 │

│ aes-256-cfb128 with IV │ 5E6CB398F653 │

│ aes-256-cbc no IV │ 1BC0629A92450D9E73A00E7D02CF4142 │

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

Пример в режиме -gcm:

Запрос:

**INSERT** **INTO** encryption\_test **VALUES**('aes-256-gcm', encrypt('aes-256-gcm', 'Secret', '12345678910121314151617181920212', 'iviviviviviviviv')), \

('aes-256-gcm with AAD', encrypt('aes-256-gcm', 'Secret', '12345678910121314151617181920212', 'iviviviviviviviv', 'aad'));

**SELECT** **comment**, hex(secret) **FROM** encryption\_test **WHERE** **comment** **LIKE** '%gcm%';

Результат:

┌─comment──────────────┬─hex(secret)──────────────────────────────────┐

│ aes-256-gcm │ A8A3CCBC6426CFEEB60E4EAE03D3E94204C1B09E0254 │

│ aes-256-gcm with AAD │ A8A3CCBC6426D9A1017A0A932322F1852260A4AD6837 │

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

###### aes\_encrypt\_mysql

Совместима с шифрованием myqsl, результат может быть расшифрован функцией [AES\_DECRYPT](https://dev.mysql.com/doc/refman/8.0/en/encryption-functions.html#function_aes-decrypt).

При одинаковых входящих значениях зашифрованный текст будет совпадать с результатом, возвращаемым функцией encrypt. Однако если key или iv длиннее, чем должны быть, aes\_encrypt\_mysql будет работать аналогично функции aes\_encrypt в MySQL: свернет ключ и проигнорирует лишнюю часть iv.

Функция поддерживает шифрофание данных следующими режимами:

* aes-128-ecb, aes-192-ecb, aes-256-ecb
* aes-128-cbc, aes-192-cbc, aes-256-cbc
* aes-128-cfb1, aes-192-cfb1, aes-256-cfb1
* aes-128-cfb8, aes-192-cfb8, aes-256-cfb8
* aes-128-cfb128, aes-192-cfb128, aes-256-cfb128
* aes-128-ofb, aes-192-ofb, aes-256-ofb

Синтаксис:

aes\_encrypt\_mysql('mode', 'plaintext', 'key' [, iv])

Параметры:

* mode – режим шифрования. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/#string).
* plaintext – текст, который будет зашифрован. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/#string).
* key – ключ шифрования. Если ключ длиннее, чем требует режим шифрования, производится специфичная для MySQL свертка ключа. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/#string).
* iv – инициализирующий вектор. Необязателен, учитываются только первые 16 байтов. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/#string).

Возвращаемое значение:

* Бинарная зашифрованная строка. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/#string).

Примеры:

При одинаковых входящих значениях результаты шифрования у функций encrypt и aes\_encrypt\_mysql совпадают.

Запрос:

**SELECT** encrypt('aes-256-cfb128', 'Secret', '12345678910121314151617181920212', 'iviviviviviviviv') = aes\_encrypt\_mysql('aes-256-cfb128', 'Secret', '12345678910121314151617181920212', 'iviviviviviviviv') **AS** ciphertexts\_equal;

Результат:

┌─ciphertexts\_equal─┐

│ 1 │

└───────────────────┘

Функция encrypt генерирует исключение, если key или iv длиннее чем нужно:

Запрос:

**SELECT** encrypt('aes-256-cfb128', 'Secret', '123456789101213141516171819202122', 'iviviviviviviviv123');

Результат:

Received exception from server (version 21.1.2):

Code: 36. DB::Exception: Received from localhost:9000. DB::Exception: Invalid key size: 33 expected 32: While processing encrypt('aes-256-cfb128', 'Secret', '123456789101213141516171819202122', 'iviviviviviviviv123').

Однако функция aes\_encrypt\_mysql в аналогичном случае возвращает результат, который может быть обработан MySQL:

Запрос:

**SELECT** hex(aes\_encrypt\_mysql('aes-256-cfb128', 'Secret', '123456789101213141516171819202122', 'iviviviviviviviv123')) **AS** ciphertext;

Результат:

┌─ciphertext───┐

│ 24E9E4966469 │

└──────────────┘

Если передать iv еще длиннее, результат останется таким же:

Запрос:

**SELECT** hex(aes\_encrypt\_mysql('aes-256-cfb128', 'Secret', '123456789101213141516171819202122', 'iviviviviviviviv123456')) **AS** ciphertext

Результат:

┌─ciphertext───┐

│ 24E9E4966469 │

└──────────────┘

Это совпадает с результатом, возвращаемым MySQL при таких же входящих значениях:

mysql> **SET** block\_encryption\_mode='aes-256-cfb128';

Query OK, 0 **rows** affected (0.00 sec)

mysql> **SELECT** aes\_encrypt('Secret', '123456789101213141516171819202122', 'iviviviviviviviv123456') **as** ciphertext;

+------------------------+

| ciphertext |

+------------------------+

| 0x24E9E4966469 |

+------------------------+

1 **row** **in** **set** (0.00 sec)

###### decrypt

Функция расшифровывает зашифрованный текст и может работать в следующих режимах:

* aes-128-ecb, aes-192-ecb, aes-256-ecb
* aes-128-cbc, aes-192-cbc, aes-256-cbc
* aes-128-cfb1, aes-192-cfb1, aes-256-cfb1
* aes-128-cfb8, aes-192-cfb8, aes-256-cfb8
* aes-128-cfb128, aes-192-cfb128, aes-256-cfb128
* aes-128-ofb, aes-192-ofb, aes-256-ofb
* aes-128-gcm, aes-192-gcm, aes-256-gcm

Синтаксис:

decrypt('mode', 'ciphertext', 'key' [, iv, aad])

Параметры:

* mode – режим шифрования. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/#string).
* ciphertext – зашифрованный текст, который будет расшифрован. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/#string).
* key – ключ шифрования. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/#string).
* iv – инициализирующий вектор. Обязателен для -gcm режимов, для остальных режимов опциональный. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/#string).
* aad – дополнительные аутентифицированные данные. Текст не будет расшифрован, если это значение неверно. Работает только с -gcm режимами. Для остальных вызовет исключение. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/#string).

Возвращаемое значение:

* Расшифрованная строка. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/#string).

Примеры:

Рассмотрим таблицу из примера для функции [encrypt](https://clickhouse.tech/docs/ru/sql-reference/functions/encryption-functions/#encrypt).

Запрос:

**SELECT** **comment**, hex(secret) **FROM** encryption\_test;

Результат:

┌─comment──────────────┬─hex(secret)──────────────────────────────────┐

│ aes-256-gcm │ A8A3CCBC6426CFEEB60E4EAE03D3E94204C1B09E0254 │

│ aes-256-gcm with AAD │ A8A3CCBC6426D9A1017A0A932322F1852260A4AD6837 │

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

┌─comment─────────────────────────────┬─hex(secret)──────────────────────┐

│ aes-256-cfb128 no IV │ B4972BDC4459 │

│ aes-256-cfb128 no IV, different key │ 2FF57C092DC9 │

│ aes-256-cfb128 with IV │ 5E6CB398F653 │

│ aes-256-cbc no IV │ 1BC0629A92450D9E73A00E7D02CF4142 │

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

Теперь попытаемся расшифровать эти данные:

Запрос:

**SELECT** **comment**, decrypt('aes-256-cfb128', secret, '12345678910121314151617181920212') **as** plaintext **FROM** encryption\_test;

Результат:

┌─comment─────────────────────────────┬─plaintext─┐

│ aes-256-cfb128 no IV │ Secret │

│ aes-256-cfb128 no IV, different key │ �4�

� │

│ aes-256-cfb128 with IV │ ���6�~ │

│aes-256-cbc no IV │ �2\*4�h3c�4w��@

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

Обратите внимание, что только часть данных была расшифрована верно. Оставшаяся часть расшифрована некорректно, так как при шифровании использовались другие значения mode, key, или iv.

###### aes\_decrypt\_mysql

Совместима с шифрованием myqsl и может расшифровать данные, зашифрованные функцией [AES\_ENCRYPT](https://dev.mysql.com/doc/refman/8.0/en/encryption-functions.html#function_aes-encrypt).

При одинаковых входящих значениях расшифрованный текст будет совпадать с результатом, возвращаемым функцией decrypt. Однако если key или iv длиннее, чем должны быть, aes\_decrypt\_mysql будет работать аналогично функции aes\_decrypt в MySQL: свернет ключ и проигнорирует лишнюю часть iv.

Функция поддерживает расшифровку данных в следующих режимах:

* aes-128-ecb, aes-192-ecb, aes-256-ecb
* aes-128-cbc, aes-192-cbc, aes-256-cbc
* aes-128-cfb1, aes-192-cfb1, aes-256-cfb1
* aes-128-cfb8, aes-192-cfb8, aes-256-cfb8
* aes-128-cfb128, aes-192-cfb128, aes-256-cfb128
* aes-128-ofb, aes-192-ofb, aes-256-ofb

Синтаксис:

aes\_decrypt\_mysql('mode', 'ciphertext', 'key' [, iv])

Параметры:

* mode – режим шифрования. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/#string).
* ciphertext – зашифрованный текст, который будет расшифрован. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/#string).
* key – ключ шифрования. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/#string).
* iv – инициализирующий вектор. Необязателен. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/#string).

Возвращаемое значение:

* Расшифрованная строка. [String](https://clickhouse.tech/docs/ru/sql-reference/data-types/string/#string).

Примеры:

Расшифруем данные, которые до этого были зашифрованы в MySQL:

mysql> **SET** block\_encryption\_mode='aes-256-cfb128';

Query OK, 0 **rows** affected (0.00 sec)

mysql> **SELECT** aes\_encrypt('Secret', '123456789101213141516171819202122', 'iviviviviviviviv123456') **as** ciphertext;

+------------------------+

| ciphertext |

+------------------------+

| 0x24E9E4966469 |

+------------------------+

1 **row** **in** **set** (0.00 sec)

Запрос:

**SELECT** aes\_decrypt\_mysql('aes-256-cfb128', unhex('24E9E4966469'), '123456789101213141516171819202122', 'iviviviviviviviv123456') **AS** plaintext;

Результат:

┌─plaintext─┐

│ Secret │

└───────────┘

##### Функции для работы с кортежами

###### tuple

Функция, позволяющая сгруппировать несколько столбцов.

Для столбцов, имеющих типы T1, T2, … возвращает кортеж типа Tuple(T1, T2, …), содержащий эти столбцы. Выполнение функции ничего не стоит.

Кортежи обычно используются как промежуточное значение в качестве аргумента операторов IN, или для создания списка формальных параметров лямбда-функций. Кортежи не могут быть записаны в таблицу.

С помощью функции реализуется оператор (x, y, …).

Синтаксис:

tuple(x, y, …)

###### tupleElement

Функция, позволяющая достать столбец из кортежа.

N – индекс столбца начиная с 1. N должно быть константой. N должно быть целым строго положительным числом не большим размера кортежа.

Выполнение функции ничего не стоит.

С помощью функции реализуется оператор x.N.

Синтаксис:

tupleElement(tuple, n)

###### untuple

Выполняет синтаксическую подстановку элементов [кортежа](https://clickhouse.tech/docs/ru/sql-reference/data-types/tuple/#tuplet1-t2) в место вызова.

Синтаксис:

untuple(x)

Чтобы пропустить некоторые столбцы в результате запроса, вы можете использовать выражение EXCEPT.

Параметры:

* x – функция tuple, столбец или кортеж элементов. [Tuple](https://clickhouse.tech/docs/ru/sql-reference/data-types/tuple/).

Возвращаемое значение

* Нет.

Примеры:

Входная таблица:

┌─key─┬─v1─┬─v2─┬─v3─┬─v4─┬─v5─┬─v6────────┐

│ 1 │ 10 │ 20 │ 40 │ 30 │ 15 │ (33,'ab') │

│ 2 │ 25 │ 65 │ 70 │ 40 │ 6 │ (44,'cd') │

│ 3 │ 57 │ 30 │ 20 │ 10 │ 5 │ (55,'ef') │

│ 4 │ 55 │ 12 │ 7 │ 80 │ 90 │ (66,'gh') │

│ 5 │ 30 │ 50 │ 70 │ 25 │ 55 │ (77,'kl') │

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

Пример использования столбца типа Tuple в качестве параметра функции untuple:

Запрос:

**SELECT** untuple(v6) **FROM** kv;

Результат:

┌─\_ut\_1─┬─\_ut\_2─┐

│ 33 │ ab │

│ 44 │ cd │

│ 55 │ ef │

│ 66 │ gh │

│ 77 │ kl │

└───────┴───────┘

Пример использования выражения EXCEPT:

Запрос:

**SELECT** untuple((\* **EXCEPT** (v2, v3),)) **FROM** kv;

Результат:

┌─key─┬─v1─┬─v4─┬─v5─┬─v6────────┐

│ 1 │ 10 │ 30 │ 15 │ (33,'ab') │

│ 2 │ 25 │ 40 │ 6 │ (44,'cd') │

│ 3 │ 57 │ 10 │ 5 │ (55,'ef') │

│ 4 │ 55 │ 80 │ 90 │ (66,'gh') │

│ 5 │ 30 │ 25 │ 55 │ (77,'kl') │

└─────┴────┴────┴────┴───────────┘

Смотрите также:

* [Tuple](https://clickhouse.tech/docs/ru/sql-reference/data-types/tuple/).